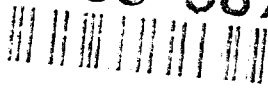


AD-A238 887



1



DTIC

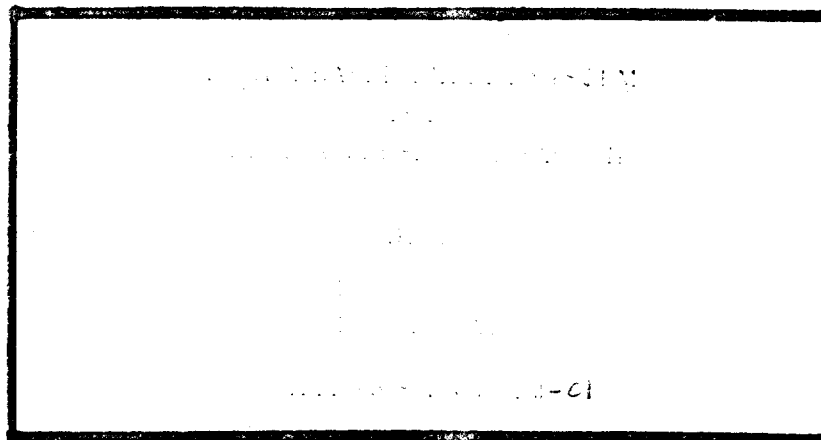
SELECTE

JUL 22 1991

S

D

D



DISTRIBUTION STATEMENT A

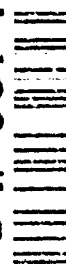
Approved for public release
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

91-05717



91 7 19 158

AFIT/GCS/ENG/91-J-01

1

DTIC
ELECTE
JUL 22 1991
S D D

AN ADA BASED EXPERT SYSTEM
FOR
THE ADA VERSION OF SATool II

THESIS

Min-fuh Shyong
Major, ROCAF

AFIT/GCS/ENG/91-J-01

91-05717



Approved for public release, distribution unlimited

REPORT DOCUMENTATION PAGE

Form Approved
GSA FPMR (41 CFR) 101-11.6

1. AGENCY USE ONLY (Leave blank) 2. REPORT DATE 3. REPORT TYPE AND DATES COVERED

6 June 1991

4. TITLE AND SUBTITLE

5. FUNDING NUMBERS

An Ada Based Expert System for the Ada Version of
SAtool II (Volume I & II)

6. AUTHOR(S)

Min-fuh Shyong

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION
REPORT NUMBER

Department of Electrical and Computer
Engineering School of Engineering AFIT
Wright-Patterson AFB, OH 45433

AFIT/GCS/ENG/91J-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING
AGENCY REPORT NUMBER

SOFTWARE ENGINEERING BRANCH (COEE)
ROME AIR DEVELOPMENT CENTER
GRIFFISS AFB NY 13411
F. Lamonica

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

Distribution Unlimited

13. ABSTRACT (Maximum 200 words)

This thesis continues the work of implementing the expert system for the Ada version of SAtool II, an software design requirement analysis tool. The background, history, design process together with the design results and validation of the implementation with Ada and CLIPS/Ada is presented.

SUBJECT TERMS

Syntax Checking Expert System, SA, SADT, SAtool,
SAtool II, ES, CLIPS, CLIPS/Ada.

NUMBER OF PAGES

252

PRICE CODE

SECURITY CLASSIFICATION

SECURITY CLASSIFICATION

SECURITY CLASSIFICATION

Unclassified

Unclassified

Unclassified

UL

AN ADA BASED EXPERT SYSTEM
FOR
THE ADA VERSION OF SAtool II

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Computer System)

Min-fuh Shyong, B.S.
Major, ROCAF

June, 1991

Accession For	
NTIS GRA&I	J
DIC TAB	L
Unannounced	
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Spec
A-1	



Preface

This thesis continues the work of implementing the expert system for the Ada version of SAtool-II , which is based on the essential data model of the IDEF₀ language (16). IDEF₀ is a graphic approach to system description developed by SofTech, Inc. for the U.S. Air Force Program for Integrated Computer-Aided Manufacturing (ICAM) and is a subset of the Structured Analysis (SA) language (24) (20). The research goal is to develop an object based Ada CASE tool (SAtool II) using the abstract data model as the requirements document (16) (28).

The essential subsystem has been developed for future integration with the Ada based, IDEF₀ CASE tool, SAtool II, (28). The original SAtool was developed by Steve Johnson (13). D. H. Jung's explored the idea of performing syntax checking on the SAtool output(14). His research focused on the prototype development of an IDEF₀ syntax (language) validation tool which is an expert system to perform a syntax validation of the IDEF₀ diagram. Intack Kim continued research on the integration of an expert system with SAtool(15). Overlapping with the work of Kim, Terry Kitchen and Jay Tevis jointly designed the essential model and graphics editor model for the Ada based SAtool.

The development of this subsystem, as well as SAtool II, is part of ongoing research at the Air Force Institute of Technology, in association with the Strategic Defense Initiative Organization (SDIO), on the use of IDEF₀ as a software requirements modeling methodology. SAtool II has shown that an Ada based expert system to check the syntax of an IDEF₀ diagram is feasible. In this thesis, we discuss the design, development, implementation, validation and results of the continuing research on the expert system. This research is performed to determine the feasibility of Ada in the development of CASE tools and expert systems and to provide a subsystem that will be integrated with SAtool II.

I would like to thank the many people who supported me during this research. First of all,

I'd like to express my gratitude to Dr. Gary B. Lamont, my thesis advisor, for his guidance and inspiration through this research.

I thank my thesis committee members, Dr. Thomas C. Hartrum and Capt. Robert J. Hammell II for their contribution to this thesis.

I would also like to thank Dr. Frank M. Brown and Major Gunsch, who were my AI instructors and had given me many advices. And Dr. Mark Roth, my course advisor who helped me through all the efforts. In addition, I thank all the professors, faculties that have helped me either directly or indirectly to accomplish my effort at AFIT.

My greatest thanks to my parent for their encouragement, and my wife, Jin-rong, whose love, devotion, and morale support kept me going through all the long days and nights.

Finally, I want to thank my daughter Chien-huey and son Jiun-yenn, and I will build them up to understand that knowledge is the source of power, so they will know why daddy is always studying.

Min-fuh Shyong

Table of Contents

	Page
Preface	ii
Table of Contents	iv
List of Figures	vii
List of Tables	viii
I. INTRODUCTION	1
Background	1
History	11
Problem Statement	13
Assumptions	13
Research Approach	15
Materials and Equipment	16
Scope and Limitations	16
Sequence of Presentation	16
II. LITERATURE REVIEW	18
Introduction	18
IDEF ₀	18
Introduction to CLIPS	29
Essential Model of the IDEF ₀ Abstract Data Model.	30
Facts utilities.	31
CLIPS Working Memory Interface and Rules File.	31
Expert Systems.	32
Integration of Expert Systems with CASE Tools.	32

	Page
SAtool with Syntax Validation.	33
Specification-Transformation Expert System (STES).	34
Visible Analyst Workbench.	35
Summary	36
III. REQUIREMENTS ANALYSIS	37
Introduction	37
Consideration of the Previous Studies	37
Facts Translator Requirements-Essential_Fact.Utilities.	38
Retrieve Essential Data Model Information.	38
Restore Essential Data Model Information.	39
CLIPS_Working_Memory_Interface.	39
Essential_IO.	39
Syntax Checking Expert System Requirements	43
Summary	45
IV. HIGH LEVEL DESIGN	46
Introduction	46
Previous Study Considerations	47
IDEF ₀ Diagram Translator	47
Retrieve Procedures.	51
Restore Procedures.	53
IDEF ₀ Syntax Expert System Components	53
The Inference Engine Selected.	53
Knowledge Base.	55
Data Base (Working Memory).	56
User Interface.	56
Test Plan	57
Summary	59

	Page
V. DETAILED DESIGN, IMPLEMENTATION, AND TESTING	61
Introduction	61
IDEF ₀ Diagram Translator Implementation	61
Expert System Syntax Checking Rules Design	62
IDEF ₀ Diagram Syntax Analysis.	64
Syntax Checking Environment.	67
Essential Model Facts Format Analysis for Boundary Arrows.	67
Translation Rules for Boundary Arrows.	69
Hierarchical Consistency Checking Rules.	76
Testing Expectations	80
Test Results Validation	81
Summary	83
VI. CONCLUSIONS AND RECOMMENDATIONS	84
Introduction	84
Conclusions	84
Recommendations	86
Boundary Single Data Item.	86
Boundary Pipeline Data Items.	86
Further IDEF ₀ Diagrams Drawing Features.	87
Bibliography	88

List of Figures

Figure	Page
1. Example of a IDEF ₀ Diagram	6
2. Basic Concept of an Expert System Function	9
3. Development of an Expert System	10
4. The Structure of a Rule-Based Expert System	11
5. Clips/Ada Visibility with Essential Subsystems	14
6. Components of a Context Diagram	20
7. Hierarchy Diagram for 'Control Elevator'	22
8. A-0 Essential Model Diagram for 'Control Elevator'	23
9. A0 Diagram for 'Control Elevator'	24
10. A1 Diagram for 'Control Elevator'	25
11. A12 Diagram for 'Control Elevator'	26
12. A2 Diagram for 'Control Elevator'	27
13. A26 Diagram for 'Control Elevator'	28
14. Module Diagram for Essential_Fact_Uilities	40
15. Module Diagram for Clips_Working_Memory_Interface	41
16. Module Diagram for Essential_IO	42
17. Flow Diagram for IDEF ₀ Diagram Translator	48
18. Essential Subsystems Relations and Visibility	63
19. A Typical Activity Box Features	64
20. Hierarchical Boundary Relations Between Parent and Child Activities	66
21. Pattern Matching: Rules and Facts	70
22. Intermediate Data Arrows Between Child Activities	73
23. Pipeline Consists of Data Arrow Relations	74
24. SAtool II Syntax Checking Rules visibility network	79

List of Tables

Table	Page
1. Object Classes Managers and Facts Format Extracted by Essential_Fact_Utilities . .	52
2. <i>if...then</i> Construct for the IDEF ₀ Syntax Checking Knowledge Base	77
3. Possible Syntax Expert System Checking Results	78

AN ADA BASED EXPERT SYSTEM FOR THE ADA VERSION OF SAtool II

I. INTRODUCTION

Background

To improve the productivity of quality software has been an objective ever since the first programmer Ada Lovelace first put quill pen to paper to program Babbage's analytic engine. Software development tools was expected by the software developers ever since. The recognition of the existence of the Software Crisis was initially revealed in the International Conference of Software Engineering at Garmisch, West Germany, in 1968 and continues today (4:1-3). In a sense, the essence of the software crisis is simply that it is much more difficult to build software systems than our intuition reflects (11).

In the systematic development and analysis of specific algorithms, especially for software development, computational complexity, is a field of study that runs in parallel with algorithmics. To consider globally the class of all algorithms that are able to solve a given problem is no doubt impossible in practice. Using algorithmics, we may prove by giving an explicit algorithm, that a certain problem can be solved in an acceptable time. Using complexity constraints, we try to find **any** algorithm that is capable of solving our problem correctly on all instances. Thus, it is manageable, applicable, and can be implemented. We may have found the most efficient algorithm possible. In this case we say that the complexity of the problem is known exactly; unfortunately, this does not happen often. An algorithm developer should seek the design that is consistent and within the complexity constraints of the particular effort (6:292).

Traditional computing technology has been able to develop powerful solutions for problems which can be clearly and completely *codified*—that is, problems that have algorithmic or closed form solutions. In practical problem solving there are many areas where such methods cannot be applied, where experts are needed to gather and interpret data and select a strategy for solving a problem. Such problems are typically poorly specified, difficult to define, heavily dependent upon rules of thumb—or heuristics. It is in these less well specified domains that expert systems can contribute (12:1-1).

As a rule of thumb, software development tools are crucial to the success of such an effort (19). Software development tools are designed to save both the time and effort of the designers. Here at AFIT, we have a tool called SAtool(15) that could be used as an requirement analysis tool during the first phase of the software development lifecycle, the requirement analysis phase. Tools should have a friendly interactive user interface, thus easy to learn and quick in application. Also they should produce acceptable results.

In order to develop a more powerful environment an expert system function is appropriate as a feature of SAtool(16). Once it is accomplished, the users can develop their requirements/specification using the software development requirement analysis tool. In particular they could perform syntax checking functions without time consuming manual checks. Some terms must be defined in order to facilitate the understanding of the development of such an expert system.

Computer Aided Software Engineering (CASE). In the process of developing computer software, a case tool is any software tool used by designer during the development of software. It involves all the tools that could be integrated together as a software working environment. CASE tools may be used during any of the development phases:

1. System Requirements
2. Software Requirements
3. Analysis

4. Program Design
5. Coding (implementation)
6. Testing
7. Operations (11:250)

Prior to the late 1970s, the most common method for representing user requirements for system development was informal narrative English (30:123). These requirements exhibited several undesirable characteristics (30:123-124): monolithic, redundant, ambiguous, and maintenance difficulties. The importance of well defined software requirements is crucial to the success of the particular project both in time and efficiency. Five important reasons are:

1. The later in the development life cycle that a software error is detected, the more expensive it will be to correct. More time will be wasted.
2. Many errors remain latent and are not detected until well after the stage at which they are made.
3. Many requirements errors are made.
4. Errors made in requirements specifications are typically incorrect facts, omissions, inconsistencies, and ambiguities.
5. Requirements errors can be detected(9:23-26).

The recognized need for an improved methodology led to the gradual transformation of informal methods into semi-formal methods that were graphic, partitioned, and minimally redundant (30:124-125). Early formalized methods included Data Flow Diagrams (DFDs), Entity-Relationship (E-R) Diagrams, DeMarco Data Structure Diagrams, Jackson Data Structure Diagrams, and Structured Analysis (SA) (24) (30:299-300). However, without automated tools to draw and validate the graphical models, the process of developing and maintaining the models

sometimes became overwhelming, especially for systems whose requirements constantly changed and were of considerable complexity. Naturally, this spurred research and development of a class of products known as Computer Aided Software Engineering (CASE) tools which support the drawing and validation process(30:128,464).

Structured Analysis (SA) and SADT. A similar yet alternative graphical modeling method to the DFD is SA developed by Ross (24) (30:299). According to Ross, the SA technique produces:

a hierarchically organized structure of separate diagrams, each of which exposes only a limited part of the subject to view, so that even very complex subjects can be understood. The structured collection of diagrams is called a *SA Model*. (24:17)

SA permits requirements to be modeled in one of two ways: data decomposition or activity (process) decomposition (24:19). SA is the basis for the development of the Structured Analysis Design Technique (SADT¹) by SofTech, Inc. SADT is described in the book *SADT: Structured Analysis and Design Technique* by Marca and McGowan. SADT is a graphical system for systems analysis and design. The SADT syntax is based upon an hierarchical set of diagrams. A diagram at one level is decomposed into several diagrams at a lower level to expose more detail as a program is developed (19).

IDEF₀. IDEF₀ is a requirements modeling technique developed by SofTech for the U.S. Air Force program for Integrated Computer-Aided Manufacturing (ICAM) (20). In fact, IDEF₀ stands for ICAM Definition Method Zero. IDEF₀ defines a subset of SA that omits the data decomposition and only permits requirements to be functionally or process modeled. The original purpose of IDEF₀ is the "representation of the functions of a manufacturing system or environment". However, IDEF₀ can also be used as a graphical language for modeling system requirements, including software systems. Functions(activities) are the basic objects of decomposition in SADT/IDEF₀.

¹SADT is a registered trademark of SofTech, Inc.

These functions represent different processes which may occur in a program. The graphical representation for a function is a "rectangular box". Data items needed by or produced by the different processes are graphically represented as "arrows". These arrows are grouped into four basic categories which help defining the interface between the different functions: inputs, outputs, controls, and mechanisms. More detailed explanations are illustrated below:

- function – A function represents a process or action, and is best identified by a name that starts with a verb. The function is viewed as transforming its inputs into outputs under the guidance of its controls.
- data item – Data or information produced by or needed by a function.
- input – An data item arrow enter the left side of the function box
- output – An data item arrow leave the right side of the box.
- control – Defines the condition or circumstances under which the transformation from input to output occurs.
- mechanism – An arrow entering the bottom of the function, indicate a means of performing the functions.
- call – A mechanism arrow exiting from the bottom of the box, indicates that the function is a shared model. That is, it is decomposed either elsewhere in the system model, or in another systems model.(20)

An example of an IDEF₀ diagram is shown in Fig 1, in which each field of Author, Project, Date, Rev, Node, and Title must be filled. Each activity must be named and numbered and must have at least one *control* (arrow entering the top) and one *output* (arrow leaving the right side of a box) Each data arrow must also be labeled. The function is viewed as a transforming its inputs (arrow entering the left side of a box) into outputs under the guidance of its controls. Each function in

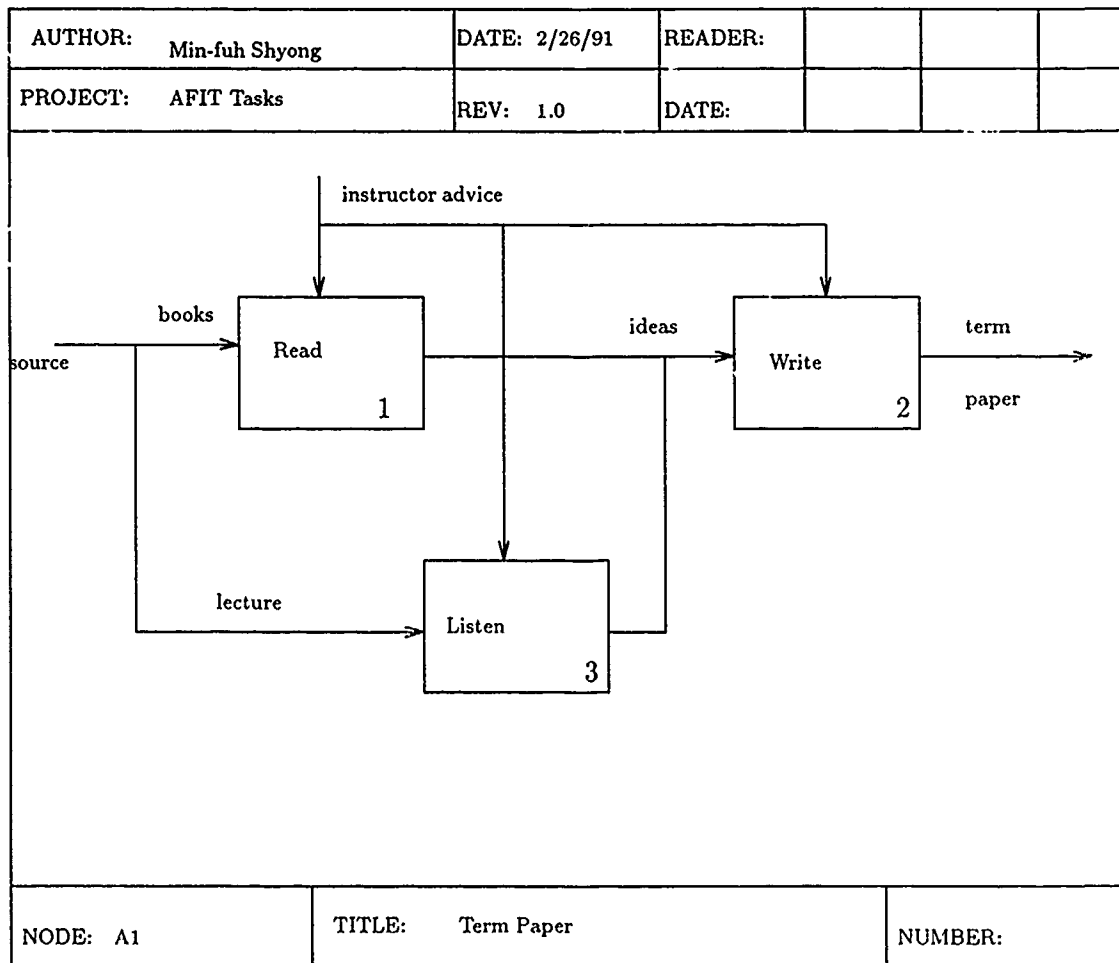


Figure 1. Example of a IDEF₀ Diagram

this diagram can be a parent diagram in the decomposition of its child diagrams. More examples and IDEF₀ syntax will be introduced in chapter 2.

Data Dictionary. The phrase data dictionary is almost self-defining. The data dictionary is an organized listing of all the data elements that are pertinent to the system, with precise, rigorous definitions so that both user and systems analyst will have a common understanding of all inputs, outputs, components of stores, and intermediate calculations (30:189). A data dictionary is a technique that usually accompanies one of the graphical modeling techniques (27:82-83).

SAtool. SAtool is a C-based CASE tool for assisting the software engineer in the requirements phase of the software development life cycle (13:6-1). SAtool's graphical language is based on IDEF₀ which, in turn, is based on the SADT. SAtool allows the user to perform requirements analysis by developing IDEF₀ diagrams and associated data dictionaries.

SAtool-II. The essential model and graphics editor model are being developed as an object based Ada CASE tool (SAtool II) using the abstract data model as the requirements document (16)(28). The development and implementation of an object oriented design (OOD) in Ada for the essential data model is achieved. SAtool-II differs from its predecessor SAtool, in that SAtool-II is to be fully implemented in Ada programming language and more functions, like a syntax checking expert system are expected to be completed in Ada as well.

SAtool-II is designed for individuals who are familiar with structured analysis and SADT. In order for SAtool-II, or any interactive analysis tool, to be effective, it must be able to capture the data information entered by the user and store it into some type of database. Thus all the input information will be examined by the tool system, prompt the user when ambiguities happen and create a complete data dictionary without further manual inputs. SAtool-II stores data derived from the SADT diagrams in a standard file format which can be read by the common database interface. The purpose of this stored standard file is for the future compatibility with as many database tools in existence as possible (17).

Expert System. The first step in solving any problem is defining the problem area or problem domain. This consideration is just as true in artificial intelligence (AI) as in conventional problem solving. According to Luger, the attempted definition of AI is:

Artificial intelligence may be defined as the branch of computer science that is concerned with the automation of intelligent behavior. AI is part of computer science and, as such, must be based on sound theoretical and applied principles of that field. These principles include the data structures used in knowledge representation, the algorithms needed to apply that knowledge, and the language and programming techniques used in their implementation.

However, this definition suffers from the fact that intelligence itself is not very well defined or understood. Thus the problem of defining artificial intelligence becomes one of defining intelligence itself. (18:1)

Because of the mystique formerly associated with AI, there is a lingering tendency to still believe the old adage **"It's an AI problem if it hasn't been solved yet"** (10:1). Professor Edward Feigenbaum of Stanford University, an early pioneer of expert systems technology, has defined an expert system as "... an computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution."

That is, an expert system is a computer system which **emulates** a small aspect of the decision-making ability of a human expert. The term emulate means that the expert system is intended to act as much as possible respects in the problem domain like a human expert. An emulation is much stronger than a simulation which is only required to act like the real thing in some respects (10:1). Internally, the expert system consists of two main components. The knowledge-base contains the knowledge with which the **inference engine** (algorithm) draws conclusions. These conclusions are the expert system's responses to the user's queries for expertise. As more knowledge is added to the intelligent assistant, it acts more like an expert (matching patterns in a logical fashion following the experts explicit reasoning that has been programmed). An expert's knowledge is specific to one **problem domain** as opposed to knowledge about general problem-solving techniques. General problem domains are medicine, finance, science or engineering and so forth in which an expert can solve specific problems very well. The expert's knowledge about solving specific problems is called the **knowledge domain** (data structure and control structure) of the expert (10:3).

Similarly, according to Luger:

An expert system is a knowledge-based program that provides "expert quality" solutions to problems in a specific domain. Generally, its knowledge is extracted from human experts in the domain and it attempts to emulate their methodology and performance. As with skilled humans, expert systems tend to be specialists, focusing on a narrow set of problems. Expert systems neither copy the structure of the human mind nor are mechanisms for general intelligence. They are practical programs that use heuristic

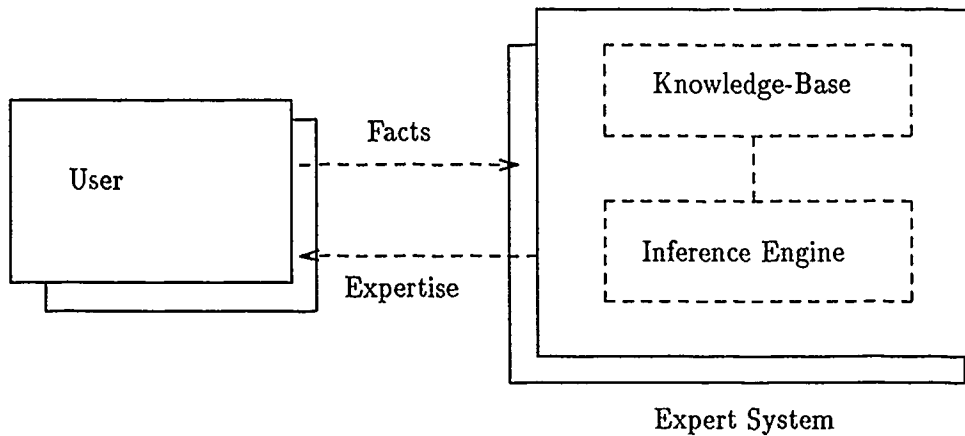


Figure 2. Basic Concept of an Expert System Function

strategies based on a certain set of algorithms developed by humans to solve specific classes of problems (18:291)(23).

A concept figure of an expert system function is shown in Figure 2.

The process of building an expert system is called **knowledge engineering**. Knowledge engineering refers to the acquisition of knowledge from a human expert or other source and to the art and science of crafting these expert systems(12:1-2). It applies to all levels of the software life cycle. But for the reasons mentioned earlier, it emphasizes the requirements phase. An expert system has the following performance characteristics:

- *High performance.* The system must be capable of responding at a level of competency equal to or **better** than an expert in the field. The term **better** means that the system will never forget things, getting tired, make mistakes, like a human expert does.
- *Adequate response time.* The system must also perform in a reasonable time, comparable to the time required by an expert to reach a decision.
- *Good reliability.* The expert system must be reliable and not prone to crashes (giving false, slow or no results) or else it will not be used.

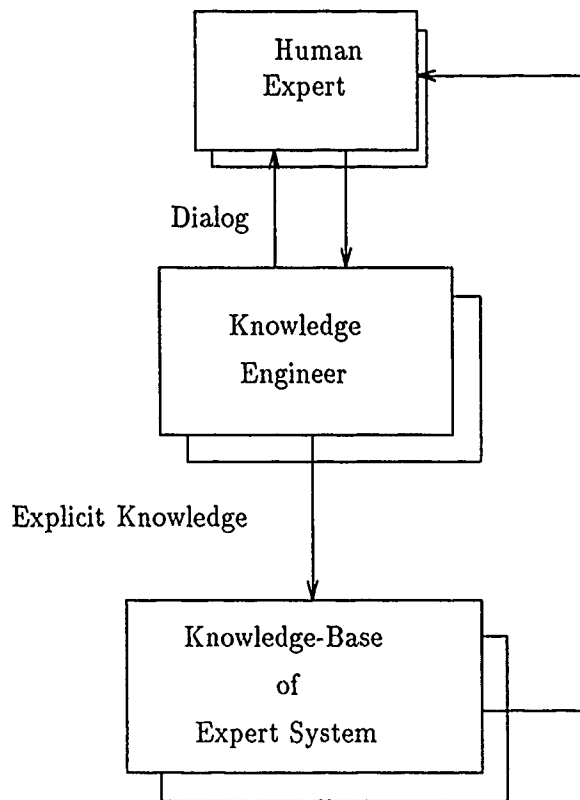


Figure 3. Development of an Expert System

- *Understandable.* The system should have an explanation capability (pattern matching) in an equivalent way that human experts can explain their reasoning. (10:6-9)

The general stages in the development of an expert system are illustrated in Figure 3.

Expert System Tools. An expert system tool is any computer language or programming system that supports the encoding of domain knowledge and provides one or more inference techniques (search-methods; select, match, act) to apply the knowledge in order to solve the problem (16:5). The structure of a Rule-Based Expert System is illustrated in Figure 4.

CLIPS (C Language Integrated Production System) is an expert system tool, CLIPS/Ada, the same tool as CLIPS but written entirely in Ada, is selected for this effort. Since CLIPS/Ada is the only expert system tool available that was written in Ada programming language. As with

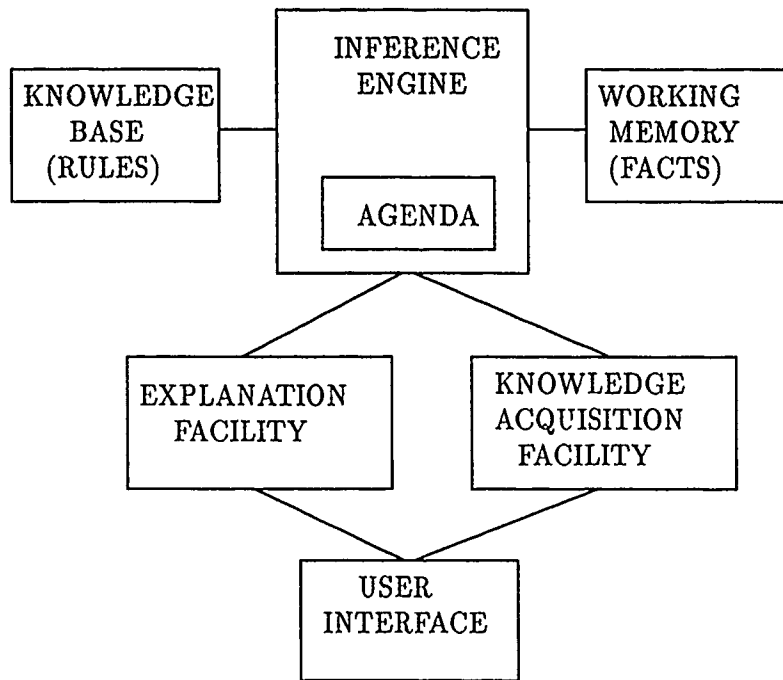


Figure 4. The Structure of a Rule-Based Expert System

most expert system shells, CLIPS/Ada already provides an inference engine and employs a forward chaining reasoning method (1:128). It is already implemented and interfaced into the SAtool II Essential Subsystem(16).

History

The original SAtool was developed by Steve Johnson (13). SAtool is an interactive computer aided software engineering (CASE) tool that permits the creation and editing of IDEF₀ diagrams based on the Structured Analysis and Design Techniques (SADT²). In fact, SAtool is sometimes referred to as a 'SADT editor'. Implemented on a Sun-3 in the C programming language, SAtool created both a graphics file and a data dictionary file; however, no syntax checking of the output was provided. The user would have to manually check the diagrams and is highly likely to neglect a mistake.

²SADT is a trademark of SofTech, Inc.

D. H. Jung explored the idea of performing syntax checking on the SAtool output(14). His research focused on the prototype development of an IDEF₀ syntax (language) validation tool which is an expert system to perform a syntax validation of the IDEF₀ diagram. The IDEF₀ syntax is formalized by converting SADT diagram constructs to predicate logic facts, and defining grammatical rules as predicates also. The research describes how both a box and an arrow are transformed to predicate logic. A C program called a *translator* translates the IDEF₀ diagram features into a formal predicate logic description that is 'readable' by the expert system. The expert system includes a backward chaining inference engine - BC3³, which uses a *goal driven* inference chain supported by Prolog-1. A chain in the goal driven inference process is a sequences of steps traversed from a hypothesis back to the facts which support the hypothesis(10:159). BC3 requires facts (knowledge) to be represented as three-element lists of the form [Object, Attribute, Value] which are normally referred to as OAV triples. Although the expert system is successful in performing the syntax validation of the IDEF₀ diagram, the rules only check a limited number of IDEF₀ features. In addition, full integration with SAtool is not achieved, since the fact file must be transferred to a separate computer, the Z-248.

Continuing research on the integration of an expert system with SAtool, Inteak Kim generated an expert system implemented in Quintus Prolog on the SUN-3(15). The entire process of IDEF₀ diagram creation, editing, and error checking is performed on the SUN-3, but the user is again required to run two separate processes: one for SAtool and one for Quintus Prolog. Even the rule base of the expert system is extended to include rules for several additional features of the IDEF₀ language, and the need for the separate microcomputer to run the expert system is also eliminated. However, fully transparent integration is still not achieved due to software compatibility problems between the C language and Quintus's version of Prolog. Besides, the data in an [O A V] tuple is limited to have only three fields introducing extra complexity in mapping the IDEF₀ syntax to the expert system rules.

³BC3 is a Prolog backward chaining expert system shell developed by F. M. Brown.

Overlapping with the work of Kim, Terry Kitchen and Jay Tevis jointly designed the essential model and graphics editor model for the Ada based SAtool. The research goal was to develop an object based Ada CASE tool (SAtool II) using the abstract data model as the requirements document (16) (28).

SAtool-II has shown that an Ada based expert system to check the syntax of an IDEF₀ diagram is feasible as a part of that system, thus the effectiveness and efficiency of the developing tool is to be evaluated. At present, a generic Ada based expert system tool (shell), CLIPS/Ada is already implemented and interfaced into the Essential Subsystem, where the subsystem is the part of SAtool II that defines the data structure of the user input data. The visibility of the CLIPS/Ada with the Essential Subsystem is illustrated in Figure 5(16).

Problem Statement

The feasibility of using an Ada based expert system has been shown in SAtool II. However, the translation of a user's hierarchical IDEF₀ diagram to a facts file is not complete, this file can be loaded into the working memory together with the CLIPS/Ada rules to check its syntax. Also, the rules for checking the syntax are incomplete. Thus, the system is not able to perform the function of checking the syntax information in a user developed hierarchical IDEF₀ diagrams with SAtool II.

This research investigation focuses on continuing the fact translation procedures, expanding SADT syntax checking rules and making the Ada based expert system of the SAtool II to a testing phase. Further investigations could be made to improve its efficiency or reevaluate its applicability to the current project.

Assumptions

Several assumptions must be made at the outset of this research.

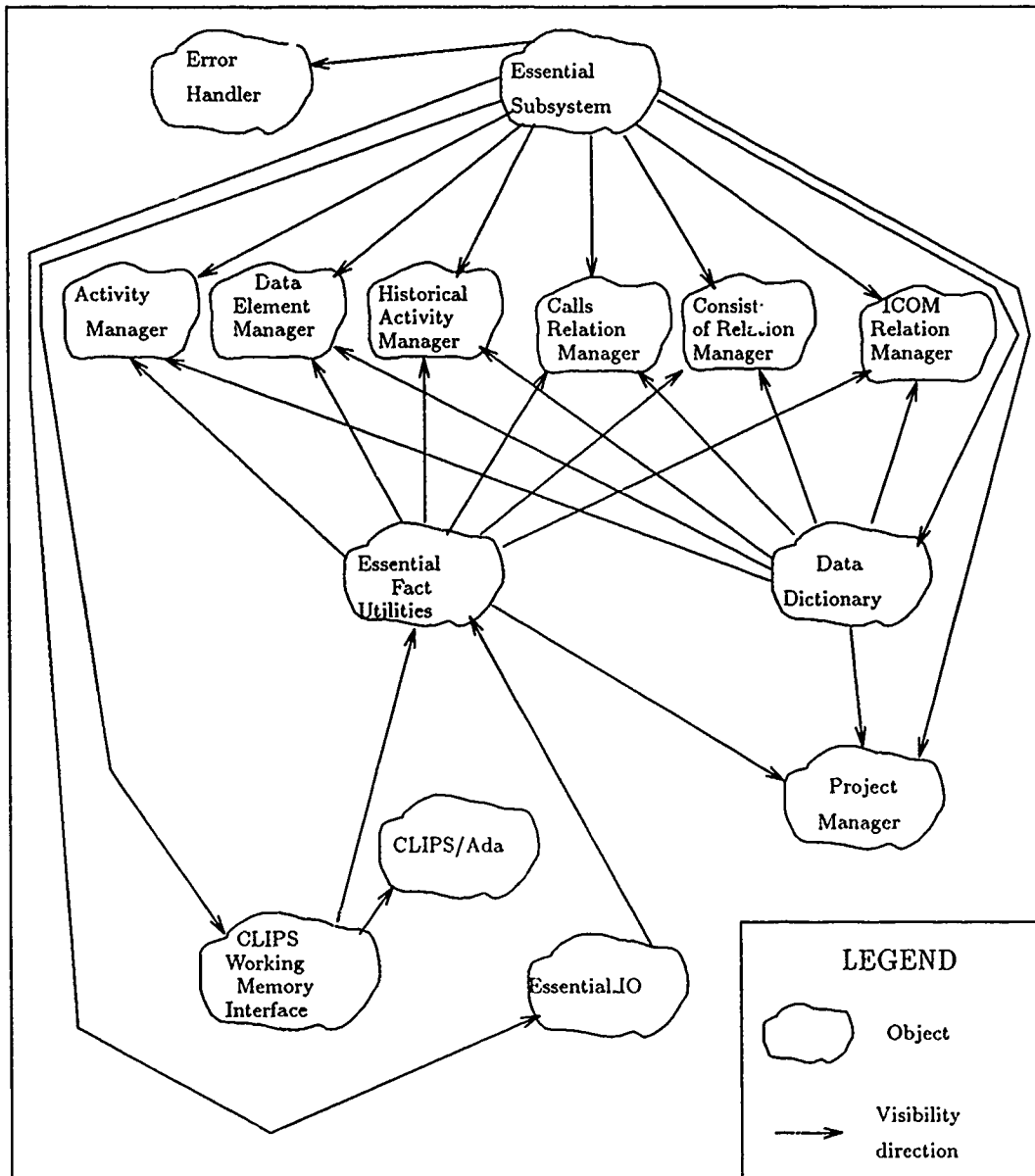


Figure 5. Clips/Ada Visibility with Essential Subsystems

1. CLIPS/Ada is an Ada version of CLIPS which has all the original functionality of CLIPS with a few exceptions(16). CLIPS/Ada has already implemented as a part of the essential subsystem as the data driven, forward chaining inference engine for the Syntax Checking Expert System.
2. Concurrent research work with the drawing data model and related SAtool II implementation issues initiated by (28) can proceed at a pace that does not hinder this concurrent research. The drawing data model is expected to implement the screen layout and drawing functions to be integrated with the essential model.
3. Users and/or researchers planning to utilize this work, must be familiar with the concepts of modeling software requirements using IDEF₀, SA, or SADT.

Research Approach

The following steps outline the intended research approach:

1. Analyze the IDEF₀ diagram syntax. "Translate" all the information that might appear in the structures including illegal ones into a facts knowledge format that can be accepted by CLIPS.
2. Complete the SAtool II Ada program to perform the functions in (1) to retrieve all the facts stored in the essential model.
3. Complete the Ada program to restore all the facts back to the essential data structure.
4. Complete the IDEF₀ CLIPS rule base that checks the facts that are translated from the users diagrams and subsequently loaded into working memory of CLIPS.
5. Demonstration programs will validate that the CLIPS expert system and fact translation procedures works.
6. Selection and use in ES application.

Materials and Equipment

The essential subsystem of SAtool II is already implemented. The CLIPS/Ada is also integrated into the subsystem. The target environment for the integration to occur is the SUN-4 workstation running a version of Berkeley Unix OS. Several workstations are readily available within the Department of Electrical and Computer Engineering to accomplish this research. The SUN-4 is the chosen platform, because it is the most readily available workstation with the X-window vs sunview graphics capability within the department.

Scope and Limitations

1. The development of subprograms within the name subsystem will translate information stored in the essential data model data structures into facts suitable for loading into the working memory of the CLIPS expert system shell.
2. The development of subprograms within the subsystem will restore all the facts from the facts file and load them back to the essential data model data structure. Thus each facts file could be separately stored, modified, and perform syntax checking in the essential model.
3. The development of an independent rules file will perform the syntax checking functions. This is to be inferenced by the CLIPS/Ada already integrated into the essential model.
4. The integration of the SAtool II subsystems with the drawing model of SAtool II.

Sequence of Presentation

This thesis is designed to be organized into 6 chapters. A short introduction to the IDEF₀ language and all the related terms are explained in chapter 1. A history of this research and its feasibility is discussed in chapter 2, literature review.

Chapter 3 presents a review of the requirements for the subsystem, the essential model specifications and data structures, the expert system rationale and examples. The facts translator

requirements and the relationship between the facts in a file with the working memory, and the facts in the working memory with the expert system rules are also presented. In addition, the syntax checking expert system requirements will be specified.

Chapter 4 presents the design of all the required subsystems and files to be implemented into the essential subsystem.

Chapter 5 illustrates the completion of the design and also includes the implementation, compiling and testing of the subsystem.

Finally, Chapter 6 presents a summary of the thesis work plus some conclusions and recommendations.

II. LITERATURE REVIEW

Introduction

The final goal of this thesis investigation is to design and implement an Ada based expert system formulation as a subsystem for checking the syntax of IDEF₀ diagrams derived from the essential model of SAtool II. Since the IDEF₀ language (20) is implemented by SAtool II, a detailed overview of the language is presented in this chapter and a hierarchy of IDEF₀ example diagrams are shown. Also the basic ideas of CLIPS is introduced. An example along with the behavior of CLIPS execution is provided in Appendix A.

The process of translating the IDEF₀ models into facts formats from their SAtool II data structure is introduced. It is initiated by (16). But most of the functions are not implemented. How those facts are to be used by the CLIPS expert system rule base is explained.

Finally, the Syntax Validation of SAtool II will be explained.

IDEF₀

The main concern supporting structured analysis (SA) is the decomposition of a complex problem into parts that can be more easily understood. This is facilitated by a hierarchical approach, called functional decomposition, in which a major problem is broken down into its major components, then each of them is in turn divided into its major pieces, and so forth. IDEF₀ syntax is a derivative of the SADT syntax and is used for software requirements analysis (20).

Although a decomposition can be based on data or process, IDEF₀ is based on the analysis of processes or *activities*. The decomposition is reflected through a series of *Function Diagrams* and corresponding *facing page text*, as shown in Figure 7 through Figure 13.

An IDEF₀ system model consists of a series of hierarchically related function diagrams, along with text descriptions and other supporting elements. The hierarchy of drawings

is formed by starting with a single function representing the system being modeled, and successively decomposing each function into its major subfunctions. Thus at any given level, a function diagram represents a single function of the next higher level, and presents the major subfunctions of that parent function, along with the interfaces between those subfunctions.(20:7)

The overall objective of using IDEF₀ diagrams is the creation of a system's software requirements *model*. Any model is an abstraction of reality, with many details omitted and only the relevant ones included. This model serves two purposes: 1. to develop a detailed understanding of the user's requirements; 2. to provide a structured documentation of the software requirements for the use in the software design stage of the life cycle.

As mentioned earlier in Chapter 1. The most important item of decomposition in IDEF₀ is a function, which is represented by a rectangular box. Since a function represents a process or action, and is identified by a name that starts with a verb. A box might be the parent of its decompositions. An IDEF₀ model of software system requirements is constructed by starting with an A-0 diagram that consists of a single box and a number of arrows. In the highest level diagram, the single box represents the entire system and might be decomposed to any level of details. Each box on a diagram may be decomposed into a diagram of its own. It is equivalent to the idea of "context diagram" as mentioned in (30:339) which is a part of the DFD modeling technique. The context diagram highlights several important characteristics of the system similar to the A-0 diagram:

- The people, organization, or systems with which the developed system communicates. These are known as terminators.
- The data that our system receives from the outside world and that must be processed in some way.
- The data produced by our system and sent to the outside world.

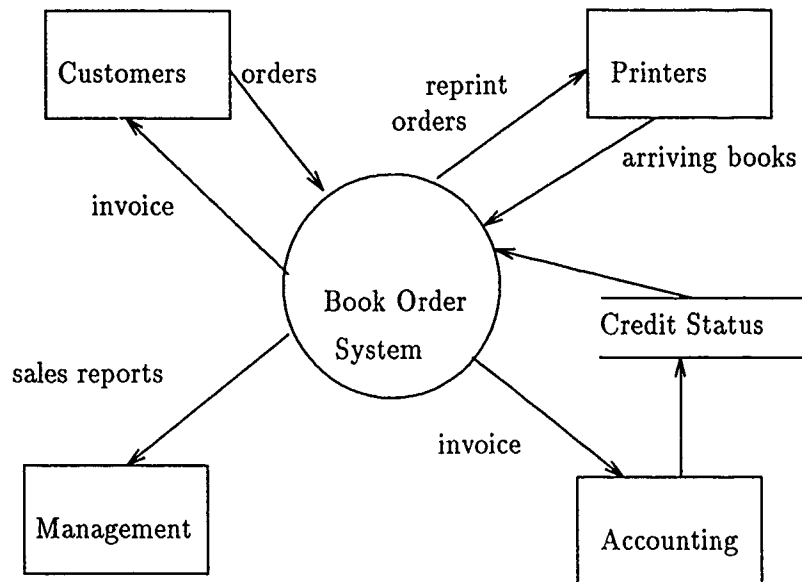


Figure 6. Components of a Context Diagram

- The data stores that are shared between our system and the terminators. These data stores are either created outside the system and used by our system or created by our system and used outside the system.
- The boundary between our system and the rest of the world.(30:339)

The concept of a **context diagram** is shown in Figure 6.

Even the idea of IDEF₀ diagram and context diagram are equivalent. Each function box in an IDEF₀ diagram must have at least one control data and one output data, the input and mechanism numbers are optional. Those are not shown in the context diagrams. The resulting functions, data manipulations of the model represented in IDEF₀ diagrams is more explicit than that of the context diagrams. Which means easier to understand and implement. Thus IDEF₀ diagrams are selected for this discussion.

In a hierarchy of IDEF₀ diagrams, Level A-0 (pronounced A minus zero) Diagram is also known as the *environment model*, as it represents the interface between what is being modeled or

analyzed and its environment. It is used to define the scope of the system. In most cases, A-0 is the highest level considered.

Since the A-0 diagram lacks the necessary detail to describe the requirements and functions of the system being developed, it must be decomposed into lower level diagrams forming a hierarchy, where each lower level in the hierarchy reveals greater detail.

Figure 7 shows the hierarchy of the example IDEF₀ model for "Control_Elevator". Therefore, each diagram in the model, with the exception of the A-0 diagram, is essentially a functional decomposition of a box in a higher level diagram. The box in the higher level diagram is appropriately called the **parent box** of the diagram.

The first actual level of decomposition is the "Level A0 Diagram," a separate drawing which represents the same level of analysis as the A-0 diagram, but which shows the major subfunctions of the system being investigated. Since it is the same level, the external interfaces on this drawing should be the same as in the A-0 diagram, in addition to the interfaces between the subfunctions. Each box on the diagram is given an integer number, beginning with "1," but the actual function numbers are "A1, A2,..." at this level. The numbers are relative to each particular box and the actual function number of a box is its integer appended to the function number of its parent. It is used to track consistency between levels.(30:24-30). Furthermore, each and every box in an IDEF₀ diagram must have at least one control arrow and one output arrow. No restrictions exist on the number of input or mechanism arrows permitted.

For more details concerning about the IDEF₀ modeling technique refer to the manual (20).

Figure 8 illustrates a single IDEF₀ diagram that represents the essential model for an elevator scheduler and controller. Figure 9 through 13 shows the decomposition diagrams simplified and translated into IDEF₀ diagrams from (30:631-652) as an example. Since the functions of creating a data dictionary has not been implemented for the essential model, so the discussion for that is omitted here.

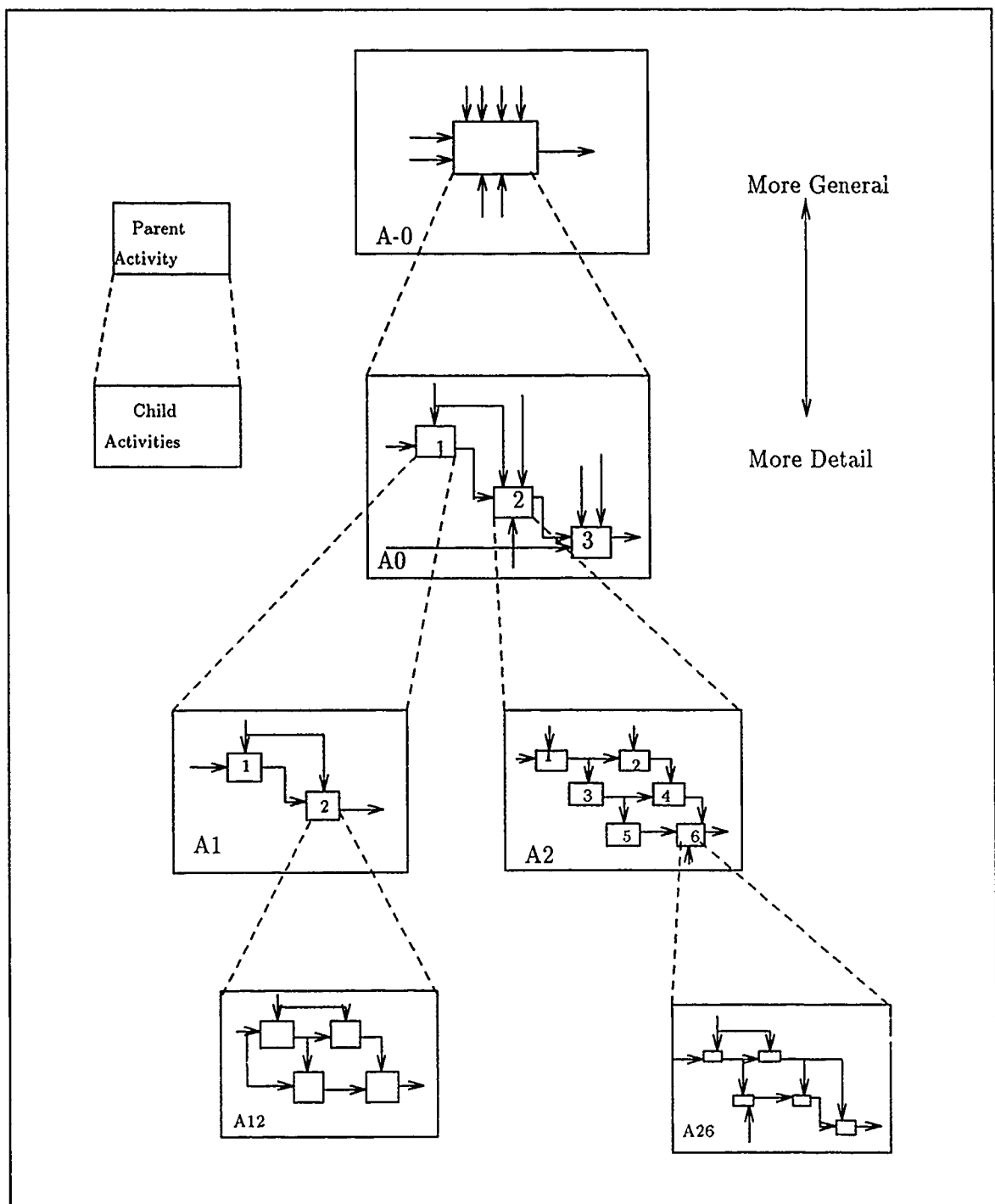


Figure 7. Hierarchy Diagram for 'Control Elevator'

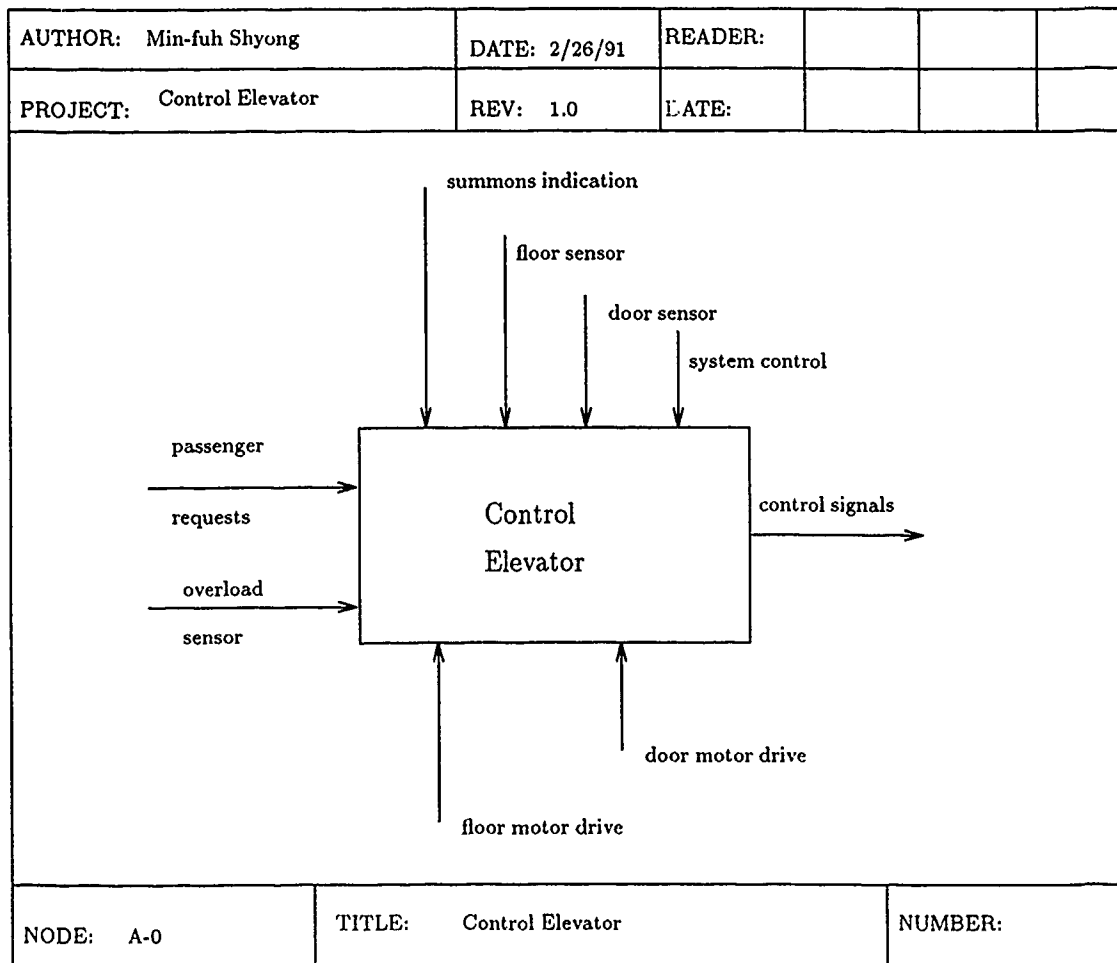


Figure 8. A-0 Essential Model Diagram for 'Control Elevator'

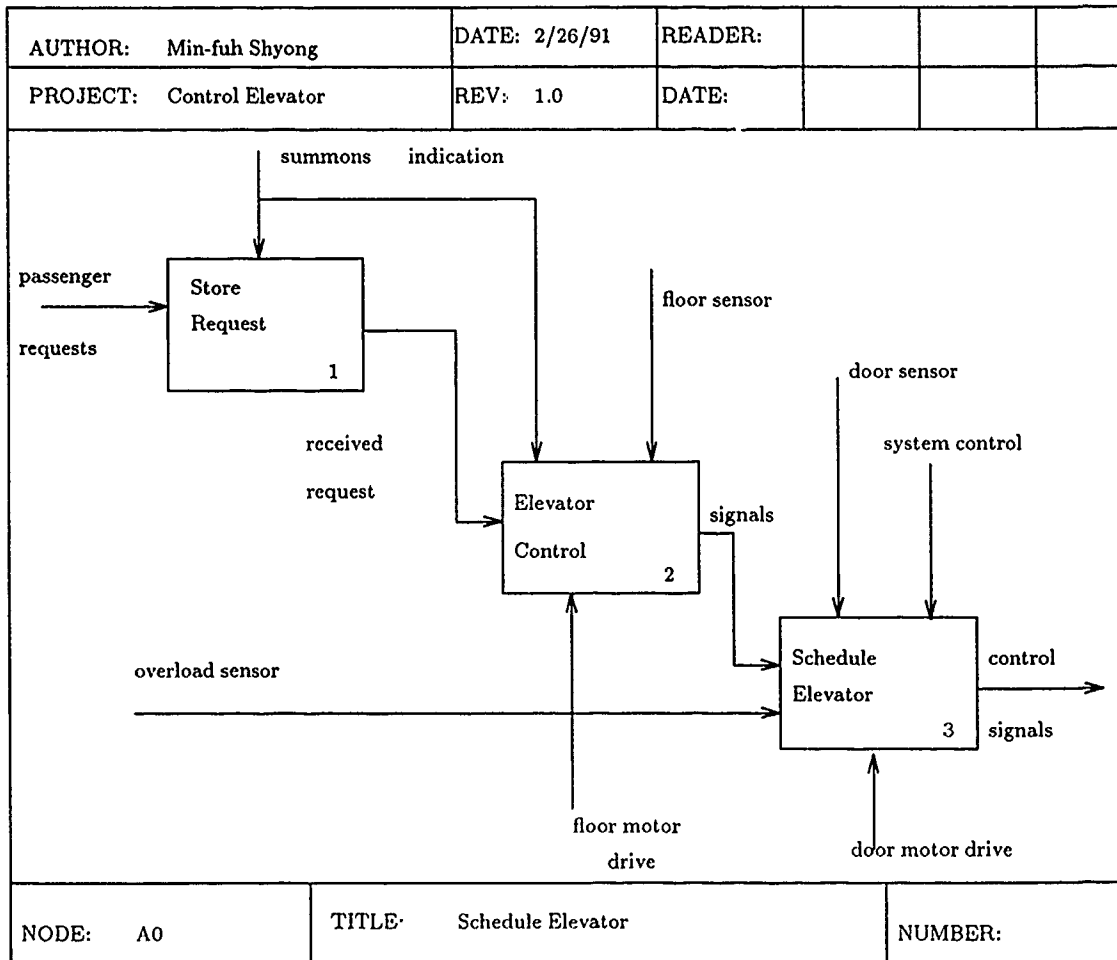


Figure 9. A0 Diagram for 'Control Elevator'

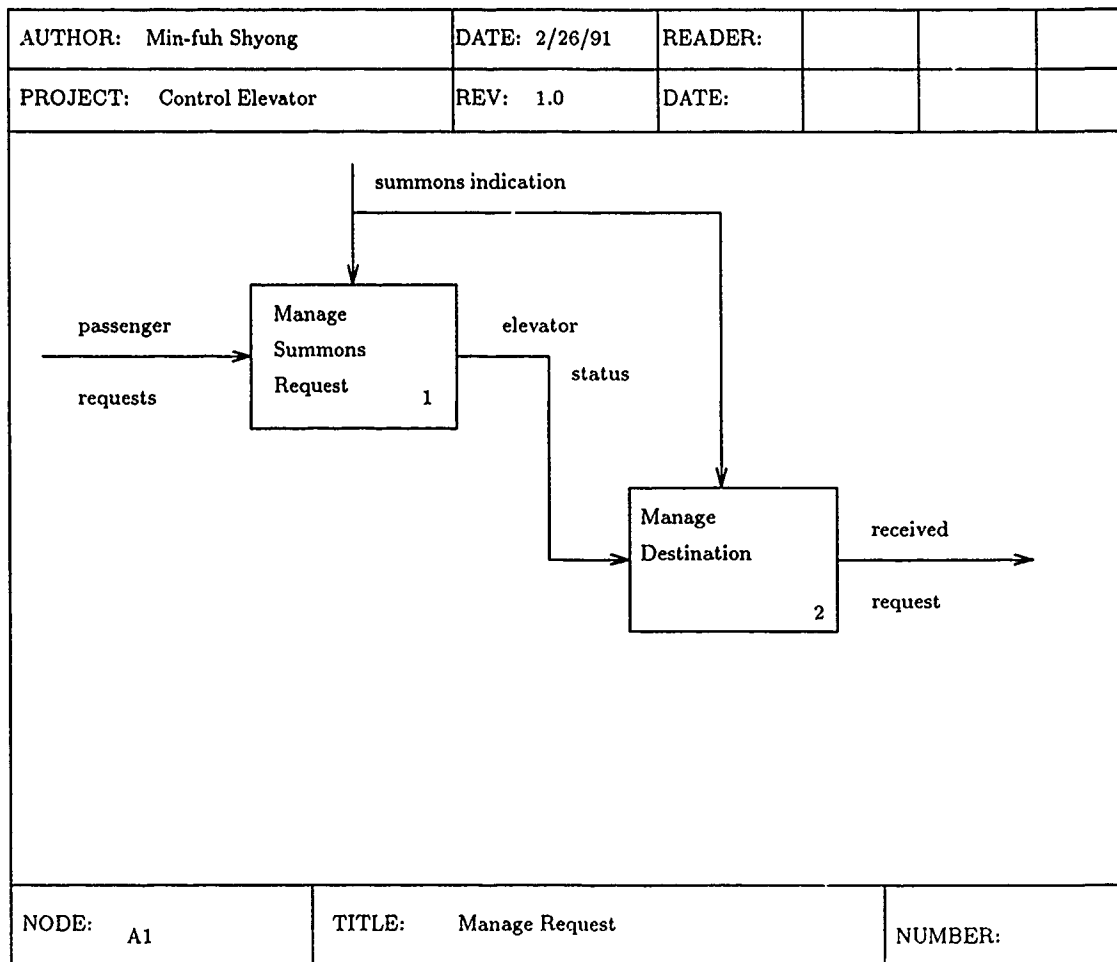


Figure 10. A1 Diagram for 'Control Elevator'

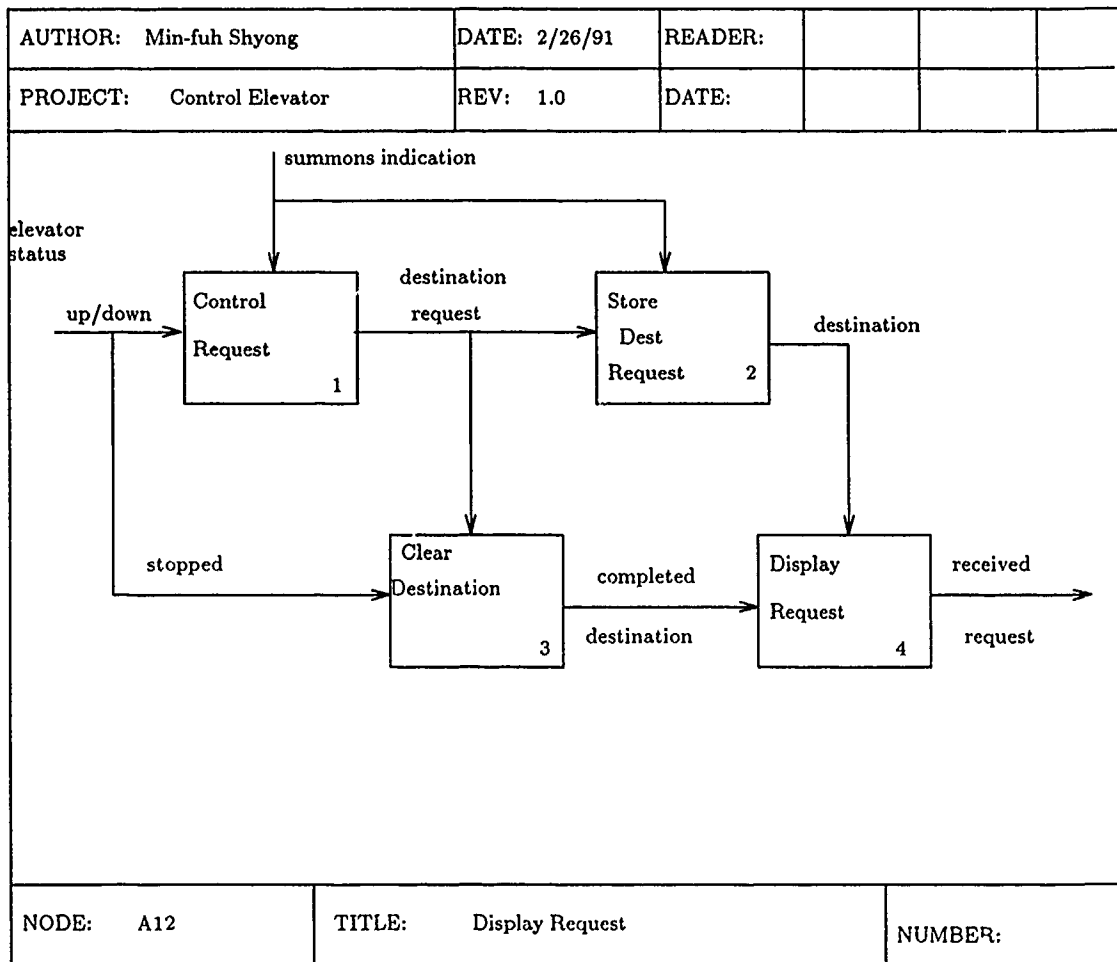


Figure 11. A12 Diagram for 'Control Elevator'

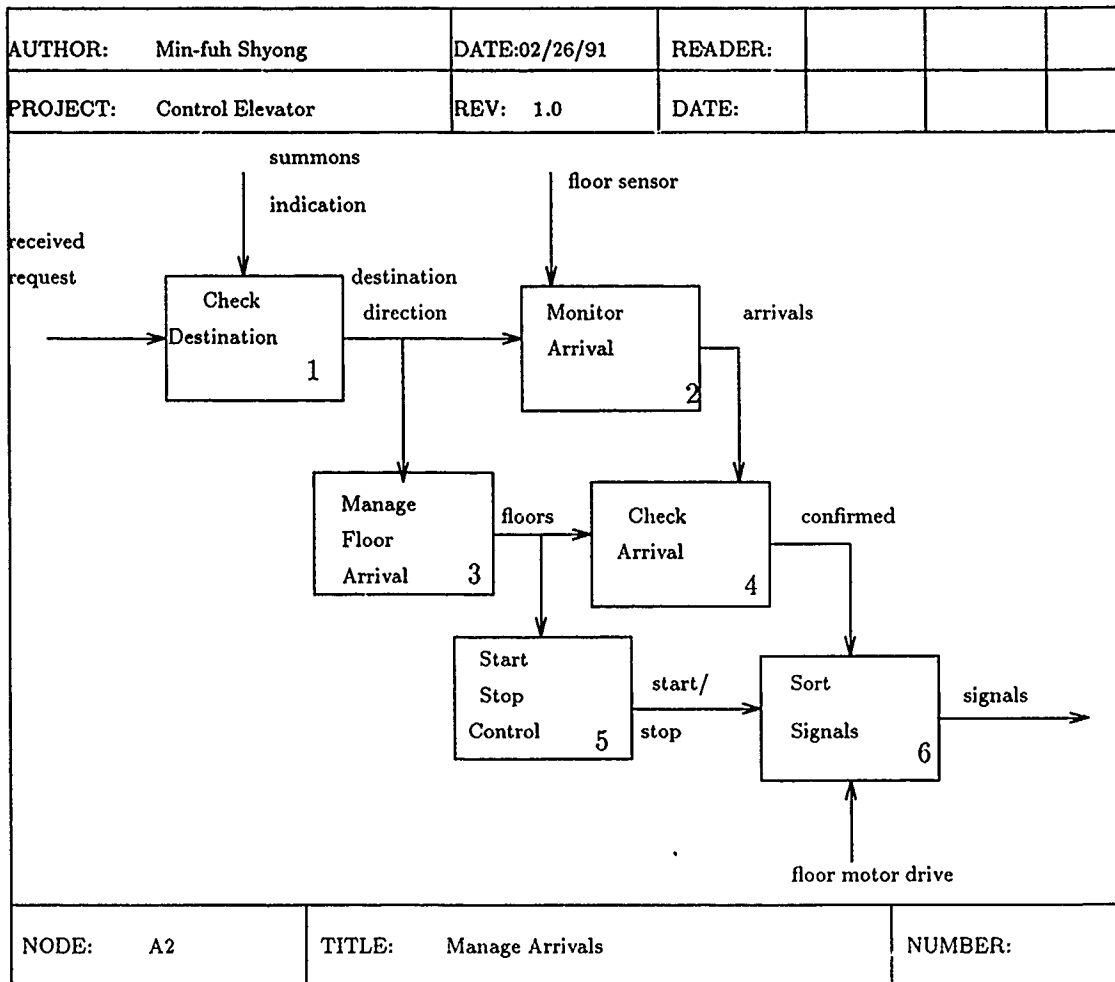


Figure 12. A2 Diagram for 'Control Elevator'

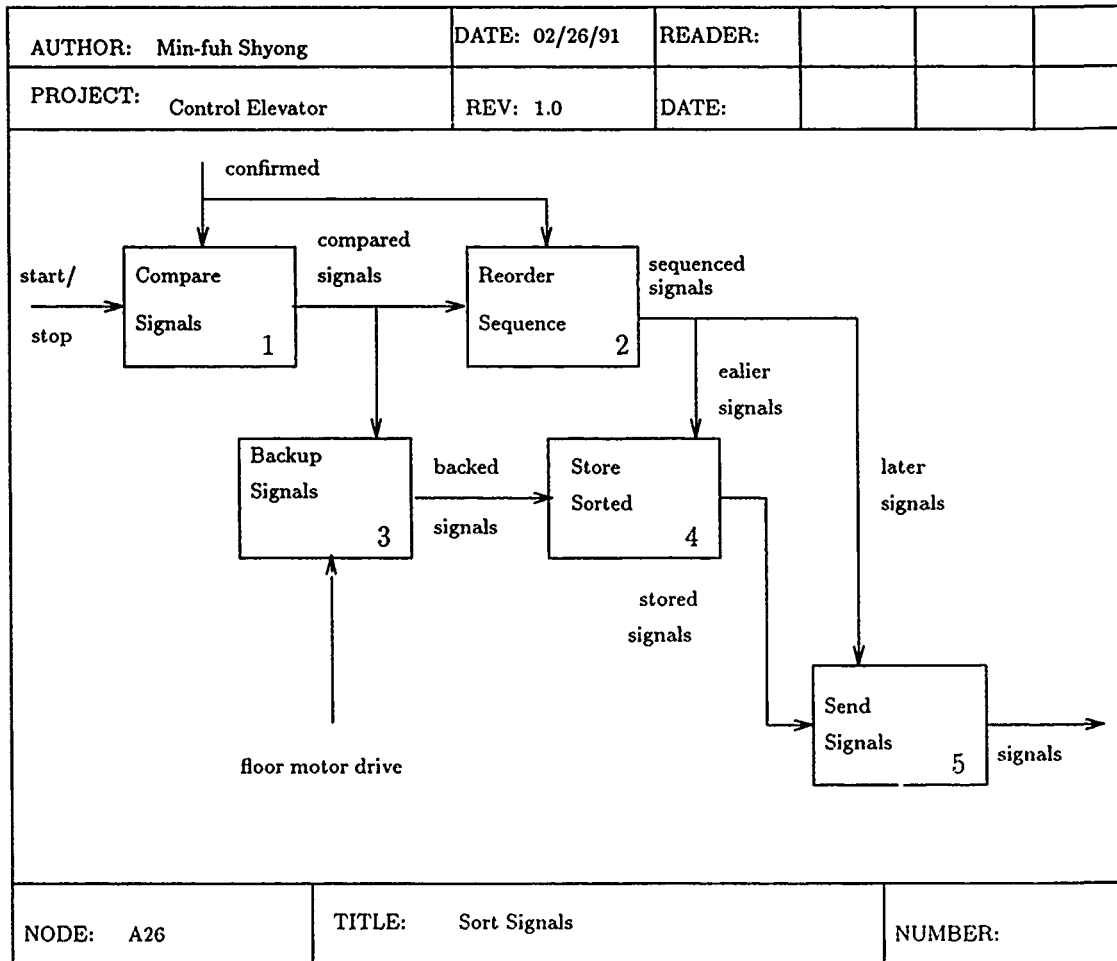


Figure 13. A26 Diagram for 'Control Elevator'

Introduction to CLIPS

To my knowledge so far, there are many practical aspects of building expert systems which must be learned by doing. Those aspects include, to say the least, the ability of defining a problem, the knowledge to approach the solution of that problem, the proficiency in using the expert system tools, the ability to reason under uncertainty, and finally the skills to implement that knowledge into an expert system. So far, building an expert system is much like writing a program in a procedural language. We have learned a lot of theories and algorithms for procedural languages at AFIT. But knowing how an algorithm works is not equivalent to being able to write a procedural program to perform that algorithm. Due to the fact of the complexity of the problem, it is not always possible for a software designer to build a system that will reflect all the intuitions the designer intended to implement. But the final product should be maximized on its functions in accordance with the understanding of the problem and the design techniques of the software developer. Similarly, capturing an expert's knowledge is not equivalent to building an expert system. For this reason, practical experience in using an expert system tool is invaluable in learning about expert systems. In addition to the inadequacies mentioned in Chapter 1 for using [O A V] triples to represent facts of the IDEF₀ model diagrams. CLIPS, which means C Language Integrated Production System is introduced here. The basic elements of CLIPS are (10:373):

1. **fact-list**: global memory for data. Each fact is a *chunk* of information in CLIPS. A fact consists of one or more fields enclosed in matching left and right parentheses.
2. **knowledge-base**: contains all the rules. Rules can be typed directly into CLIPS or they can be loaded in from a file of rules created by an editor. Each rule is of the form:

```
(defrule name-of-the-rule
  LHS-of-the-rule (conditions)
  =>
  RHS-of-the-rule (actions))
```

Where LHS
=>
RHS logically implies:

if LHS(Conditions)... THEN RHS(Actions).... All the LHS conditions are logically and'ed together.

3. **inference engine:** controls overall execution, applying all the rules to the fact-list and derive expert results. It can be a combinatorial search process.

CLIPS is a **forward chaining** rule-based language that has inferencing, pattern matching, state searching, and representation capabilities. The design of CLIPS is such that rules only match facts that have been entered after the rules. Thus newly entered rules will not match the facts that are currently on the fact-list. Only new facts that are entered will be seen by the rule. This means that a rule can only be activated by facts that are asserted after the rule is entered, thus during the execution, after new facts are asserted as a result of some rules firing, then it might activate existing rules that were not matched before (10:373-387).

Essential Model of the IDEF₀ Abstract Data Model.

The subsystems of the essential model which are already partially implemented but not complete include:

- The necessary data structures to hold the essential data model state information.
- The storing and restoring of essential data model state information which is in the form of CLIPS/Ada facts.
- the capability to create one or more data dictionary entries.
- The capability to check the syntax of an IDEF₀ model by means of a rule base consisting of IDEF₀ syntactical checking rules(16).

Facts utilities. The storing and restoring of essential data model state information in the form of CLIPS/Ada facts is performed by a separate file *Essential_Fact_Uilities*. The file is an "iterator" performed by a set of operations that iterates through the data structure of the essential model(3:52-62).

Through precisely defined formats of the output file, the informations from the entire essential data model (i.e., the project) is retrieved and stored as a CLIPS/Ada readable file format:

(attribute object value, value,...)

Where attribute defines the type of fact, object is the actual name of the data, and values are a set of descriptions of the object with unlimited length but each value must be separated by one or more spaces; i.e., (project-name Project_Name) where capitalized identifiers in the parenthesized fact file represents information derived for the data structure of the essential model. Since each project is stored in the form of facts file. A set of restore functions is also needed to restore the facts format back into the IDEF₀ essential data model, thus we don't need to regenerate the model again.

Of particular interest is the two different groups of facts: syntactical facts, and state representation facts. Syntactical facts are those sent to the CLIPS working memory for syntax checking, whereas state representation facts simply represent the state of the IDEF₀ model. The reason for this is that the syntax checking rules does not need to know the detail of some data value, simply to check if it is there.

CLIPS Working Memory Interface and Rules File. The CLIPS Working Memory Interface provides the interface for the Essential Subsystem to the CLIPS/Ada expert system. It is the only package in SATool II that has visibility to the CLIPS/Ada expert system operations. When the check syntax option is selected by the user in the essential model, the facts file and another separate rules file are loaded into CLIPS Working Memory and CLIPS/Ada initiates the logical sequence

to show the syntactical checking results of the created IDEF₀ model. It is an interface designed by (16).

The rules file is to be expanded and tested as an integral part of the essential model.

Expert Systems.

In the process of building an expert rule system using an expert system shell, we are following a step by step method of building a program(10:419).

- First: pseudo rules were written using English-like text.
- Second: the pseudo rules were used to determine the types of facts that would be required. Templates describing the facts were designed, and the initial knowledge (facts) was coded using these templates.
- Finally: the pseudo rules were translated to CLIPS rules using the fact templates as a guide for translation.

To show how this works, an example from (10:413–419) was introduced and its behavior during CLIPS execution is shown as explained in Appendix A: CLIPS BEHAVIOR IN THE BLOCKS WORLD.

Integration of Expert Systems with CASE Tools.

Through examining and understanding the projects developed from different perspectives, the strengths and weakness of each project can be identified for future reference. Today, expert systems are built on a variety of software and hardware platforms. Because of these various platforms, AFIT, academia, and industry have begun both theoretical and actual development of systems that integrate CASE tools with expert system. Each of the following projects have been implemented with differing degrees of success.

SAtool with Syntax Validation.

As mentioned previously, SAtool (13) is simply a graphical editor and provides no advice or assistance to the user as the IDEF₀ diagrams are being drawn. In other words, the user of SAtool could not determine if the finished diagram is consistent with the IDEF₀ graphical language except by tedious and time consuming manual inspection. To improve this, Jung initialized the idea of syntax checking ability with SAtool in his MS thesis in 1988 (14) His research focused on the prototype development of an IDEF₀ syntax (language) validation tool which is an expert system to perform a syntax validation of an IDEF₀ diagram. The IDEF₀ syntax is formalized by converting the syntax to predicate logic facts. The research describes how both a box and an arrow are described in predicate logic.

The graphical feature BOX is translated into the predicate BOX(x), which means: x is a BOX. In the case of the ARROW, it is translated into the predicate ARROW(x), which means: x is an ARROW (16:28)

There are two steps to the syntax validation tool (16:28) First, a C program was developed called a *translator* to translate the IDEF₀ diagram features into a formal description that is 'readable' by the expert system. The expert system is a *backward chaining* expert system-BC3¹, however, required facts had to be represented as three-element lists of the form [Object, Attribute, Value] which are normally referred to as OAV triples. The IDEF₀ diagram representation is stored in multiple C data structures, and the translator program creates a file of facts based on the information in those structures (16:28)

The second and final step of the syntax validation tool is the *syntax checker* (16:28) The syntax checker's purpose is to check the IDEF₀ diagram (now represented as OAV triples) for syntactical errors. the syntax checker is, in essence, the expert system. Syntax rules such as "Each box must have a name" and "Each arrow must have a label" are converted to *if....then* constructs

¹ BC3 is a Prolog backward chaining expert system shell developed by F.M. Brown.

in a form acceptable to BC3. The syntax checker, when executed, produces error messages for the designer to review and take corrective action.

The research, however, was limited in scope. All the features of an IDEF₀ diagram are not addressed. Plus, a transparent integration of SAtool and the syntax validation tool was not achieved (i.e., a manual step remained).

Both the aforementioned integration problem as well as expanding the syntactical checks that the expert system performed were resolved(16:29). The number of IDEF₀ syntactical features checked by the expert system are expanded (16:29). To resolve the integration problem, an attempt was made to integrate SAtool with a Quintus Prolog implementation of the syntax validator (the expert system). The "new" syntax validator is simply the expert system shell BC3 with changes necessary for it to run under Quintus Prolog. Unfortunately, compatibility problems between Quintus Prolog and the C programming language result in a failure to achieve a transparent integration of the expert system with SAtool (16:29).

Currently, a more capable CASE tool, SAtool II is being developed in the Ada programming language and includes an Ada based expert system using CLIPS/Ada. Terry and Jay continued the effort of developing the Essential model and Drawing Model of SAtool II (16) (28) As mentioned earlier, the Essential Model designed and implemented by Terry needs to be expanded to complete its expert system functions.

Specification-Transformation Expert System (STES). At the University of Illinois, Tsai and Ridge have developed the Specification-Transformation Expert System (STES) which is an expert system that they have integrated with the CASE tool Teamwork developed by Cadre Technologies (29:34). Teamwork is used to create DFDs. In addition, Teamwork runs on an Apollo workstation platform and includes a built-in Access tool which allows users to access the underlying data structures that contain the DFD description. In this case, a C++ program was written by Tsai

and Ridge to access the DFD description (29:34). By implementing the STES in OPS5, which can also run on Apollo workstations, transparent integration of Teamwork and STES is achieved.

After the requirements analysis phase of the software development life cycle is completed, STES can be used in the next step — the design phase. STES assists the software engineer with the design phase by transforming the DFD into a structure chart (16:2). The STES is used to examine the C++ representation of the DFDs, extracts the salient features, and converts them into production rules. The STES then “applies inference to identify and transform the efferent, afferent, and transformation-centered components of the dataflow diagram into a first-cut structure chart” (16:29–30).

Visible Analyst Workbench. Visible Analyst Workbench ² is an IBM-PC based CASE tool marketed by Visible Systems Corporation that contains *rules* to perform error checking of DFDs (16). According to the product documentation, the CASE tool portion called Visible Analyst allows the user the choice of two different styles in DFD construction: the Yourdon/DeMarco Method ³ DFD or the Gane and Sarson Method DFD (16:29–30). Unlimited levels of DFD process decomposition are also supported. Regardless of the style chosen, however, the rules portion of the tool called *Visible Rules* can check the diagram for proper balancing, naming conventions, etc (16).

The Visible Rules are executed without leaving the DFD which means transparent integration between the CASE tool portion and the “expert system” portion is achieved. Although the word *rules* implies a rule-based expert system is used, the proprietary nature of the product does not permit the disclosure of whether the rules are implemented algorithmically or by an expert system paradigm.

²Visible Analyst is a registered trademark of Visible Systems Corporation.

³The correct reference should probably be YSM 1.0 (5).

Summary

This chapter provides a review of several subject matter areas that directly relate to this investigation. More detailed IDEF₀ syntax was explained. Also the basic structure of CLIPS was stated. The Essential Model of SAtool II is described along with its subsystems to gain insight into an expert system functions. An actual example is provided in Appendix A to show the execution of CLIPS program. Thus a user of SAtool II unfamiliar with CLIPS execution might be able to understand the behavior of the expert system through the study of this example. Presented are the format of facts and rules and the behavior of applying rules on those facts in the working memory during execution.

Several examples concerning the integration of CASE tools with expert systems are also reviewed, since this research calls for a similar integration. Clearly, all attempts at integration do not succeed. To improve the chances of successful integration, information from successful projects should be obtained and used as a foundation for further research. The integration of IDEF₀ syntax checking capabilities in SAtool II using an expert system is the key concern of this research.

III. REQUIREMENTS ANALYSIS

Introduction

This chapter presents a review of the requirements for the subsystem to be integrated with SAtool II. First, the *IDEF₀ Diagram Translator* is implemented as a separate package and Essential Fact Utilities is used to translate any IDEF₀ diagrams drawn by SAtool II into a set of CLIPS readable facts format. The second category is to design and implement the *IDEF₀ Syntax Expert System* which is to be an application of a knowledge-based expert system. It is also a separate file having access only to the CLIPS working memory in the essential model.

This chapter presents the considerations related to the development of the IDEF₀ Diagram Translator requirements, IDEF₀ Syntax Expert System requirements, formalization criteria, the expected results, and validation test requirements.

Consideration of the Previous Studies

As mentioned in (15), the syntax checking ability was provided to find any inconsistencies for *boundary arrows* with the parent IDEF₀ diagram. But as mentioned earlier, Object, Attribute, Values data structure were used to represent the IDEF₀ diagrams known as OAV triples. Also, compatibility problems between Quintus Prolog for syntax checking, and the C programming language for IDEF₀ diagrams translator, resulted in a failure to achieve the transparent integration of the expert system with SAtool. Thus the tool developed in (15) must currently run the systems separately. This means the facts translated from the IDEF₀ diagrams must be generated and then the inference process could begin for the syntax checking abilities.

Here, our purpose is to develop a system based on Ada. The translator is to be written in Ada and the expert system tool is also to be in Ada, CLIPS/Ada. Once the subsystems are integrated into a whole, the system should provide a CASE tool environment for SAtool II.

Facts Translator Requirements-Essential_Fact_Uutilities.

Six different mechanisms or *manager mechanisms* are already implemented in the essential model for the seven different object classes. The six manager mechanisms are:

1. Activity_Manager
2. Data_Element_Manager
3. Consists_Of_Relation_Manager
4. Historical_Activity_Manager
5. Calls_Relation_Manager
6. ICOM_Relation_Manager (16:64)

The IDEF₀ Diagram Translator is implemented as a separate file to be completed and integrated with the Essential Subsystem. It must have the following two functions:

- Retrieve procedures
- Restore procedures

Retrieve Essential Data Model Information. The information that is stored in the manager abstract state machine represents the essential part of an IDEF₀ model. This information must be extracted from the manager for output to a file or for input to the CLIPS/Ada working memory for syntax checking. This is accomplished by a package containing a series of *Retrieve (Activity, Data_Element, ICOM_Relation) Facts* procedures. Those procedures first examines the 'Type Facts Flag'. Based on the flag setting (T or F), the procedure retrieves one of two different sets of facts and inserts them into a Fact_Manager which is an instance of the Fact_Buffer_Package. If the flag is true, only facts for the expert system are inserted in the Fact_Manager. If the flag is false, only the facts necessary to permanently store the state of the essential data model (i.e., the IDEF₀ model) are inserted into the Fact_Manager. This procedure is invoked by a client program

whenever the user saves the project he/she is working on, or when the user wishes to check the syntax of the project (i.e., the current IDEF₀ model) The required format for input to the CLIPS working memory is CLIPS facts. The formats are strictly defined in the procedure through Ada string type definition in accordance with the data type defined in the aforementioned six object class managers (16).

Restore Essential Data Model Information. After each Retrieve procedure, the Restore procedure for the same object class accepts as input a buffer of facts representing state information. These facts are then *Restored* into the object class manager by this procedure. This procedure is normally executed as one of a sequence of events in the initialization of SAtool II when a previous project is loaded from disk. Thus, the user could get all his work back into the Essential Model for further rechecking or modification without having to retype everything.

The Module diagram for Essential_Fact_Uilities is illustrated in Figure 14 (16:121).

CLIPS_Working_Memory_Interface.

This package provides the interface from the Essential Subsystem to the CLIPS/Ada expert system shell. It is the only package in SAtool II that has visibility to the CLIPS/Ada expert system operations. Once all the Retrieve procedures are completed, they will be included in the interface package to be retrieved by the Fact_Buffer.Package.

The Module diagram for Clips_Working_Memory_Interface is illustrated in Figure 15 (16:120).

Essential_IO.

This is a package that necessary for the operation of SAtool II to store essential data model in a file and to load essential data model information from a file into the managers. Within the scope of this thesis effort, only the retrieve procedures for those seven object classes are to be added, facts

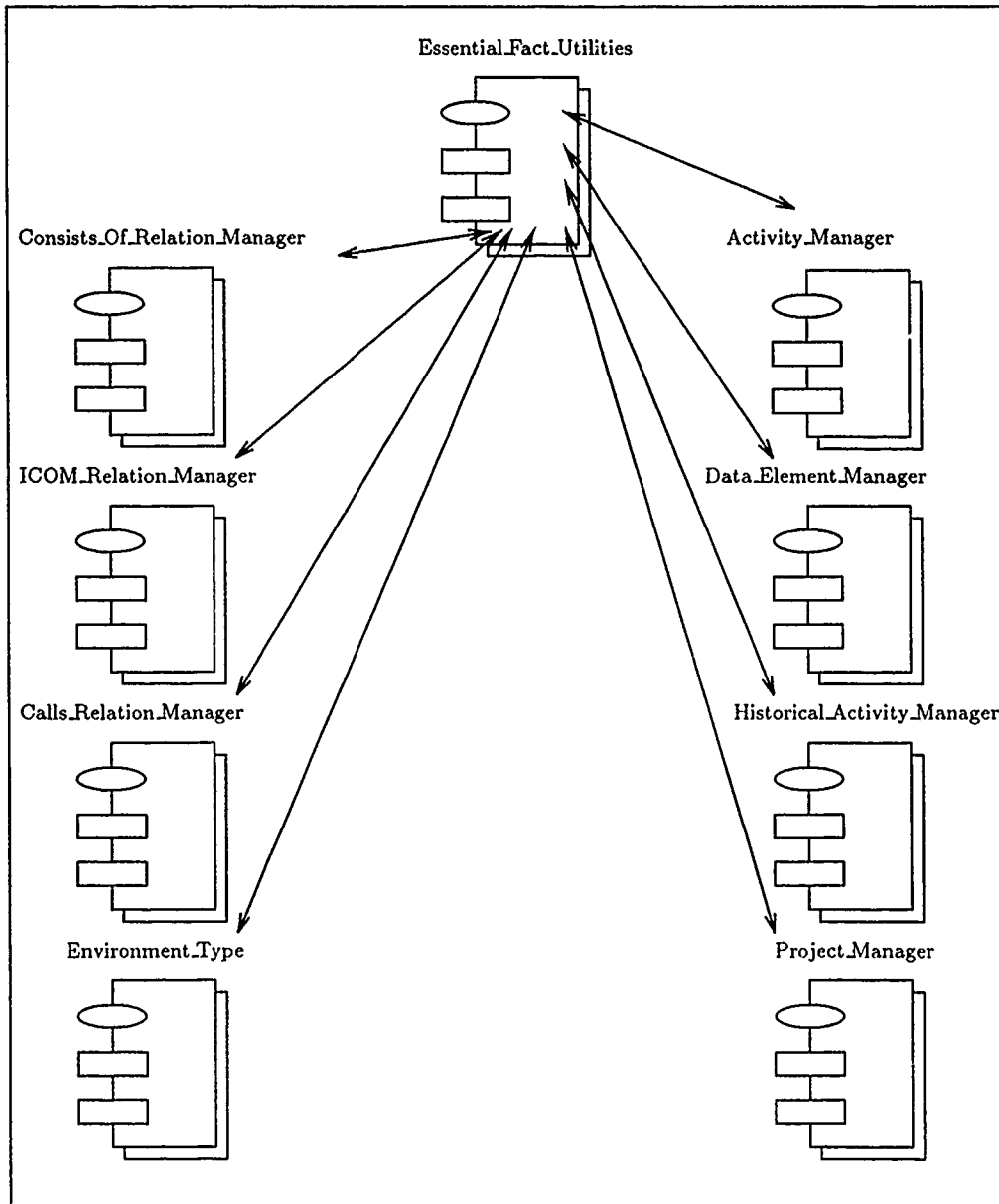


Figure 14. Module Diagram for `Essential_Fact.Utilities`

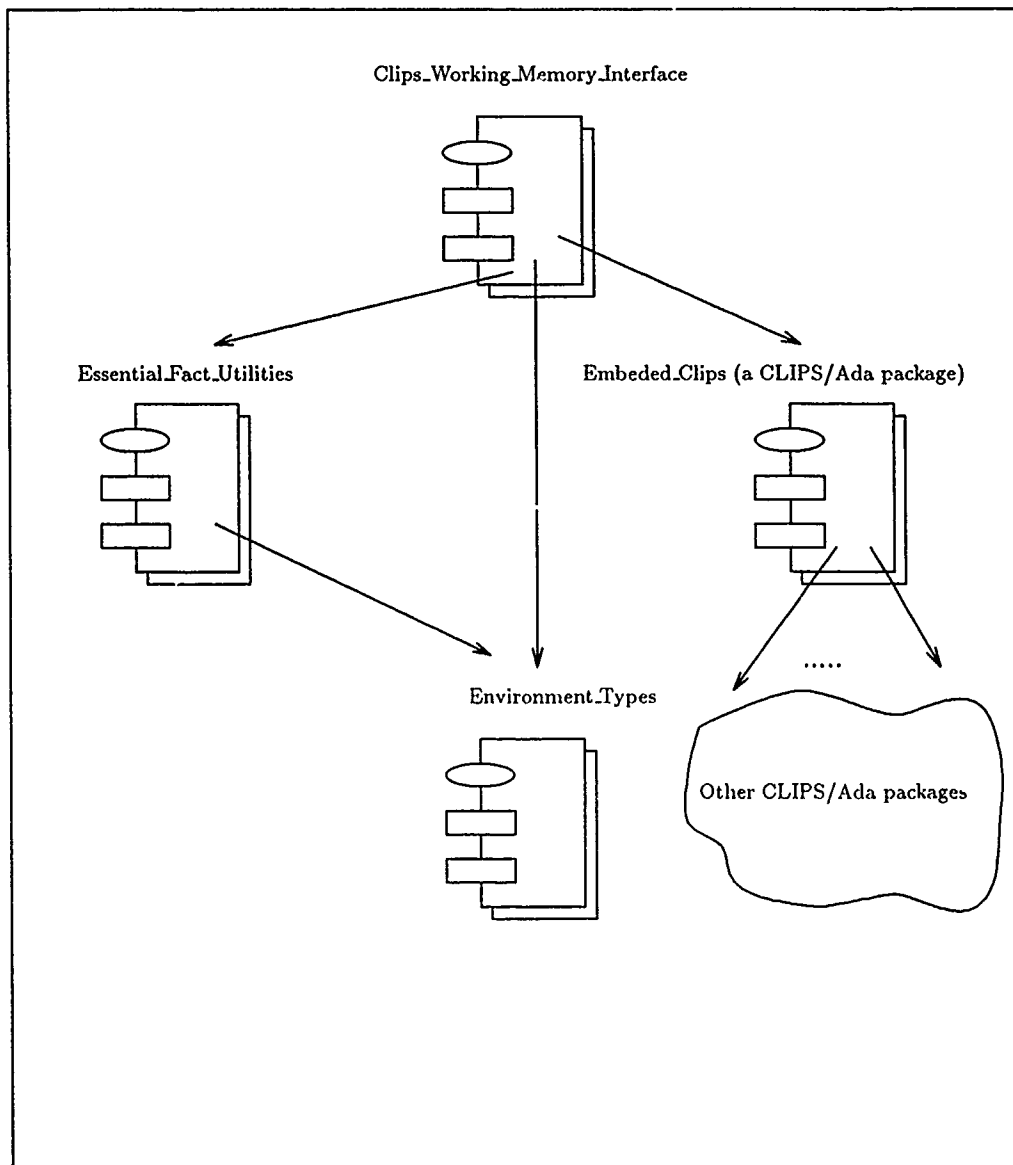


Figure 15. Module Diagram for `Clips_Working_Memory_Interface`

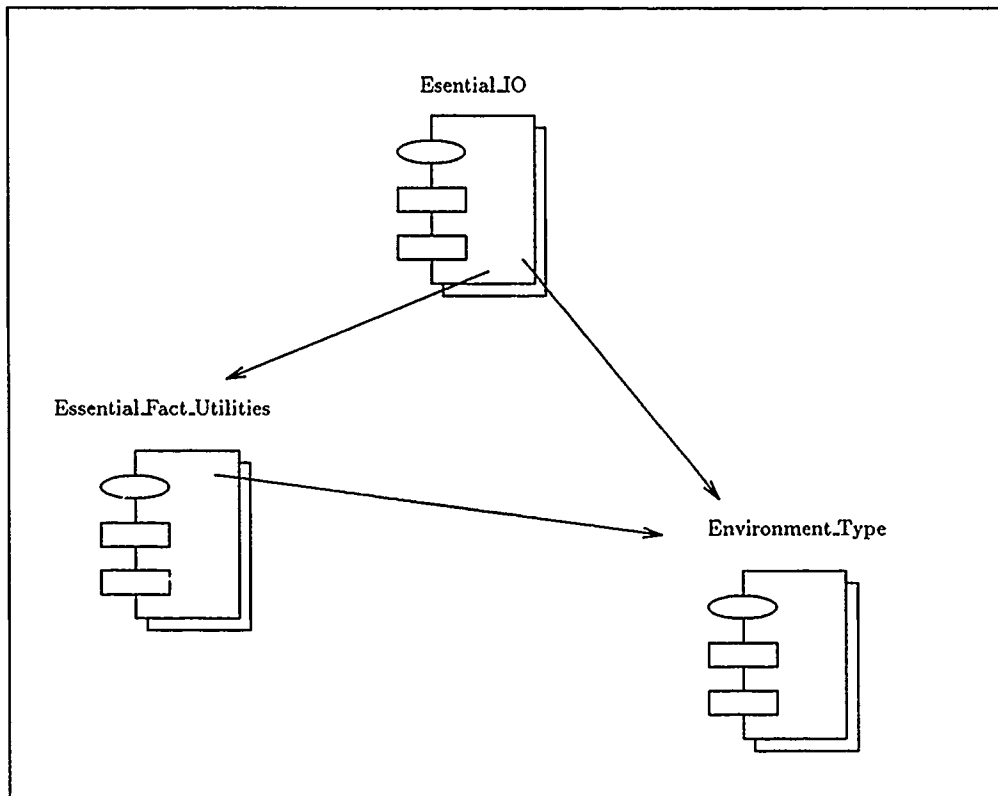


Figure 16. Module Diagram for Essential_IO

created from each object are given a fact name for for both the expert system and the Essential Model. For instance, Retrieve_ICOM_Facts in the Essential_IO package will create a facts file as:

```

(deffacts icom-facts
  (icom-attribute Name value value value)
  (
    .
    .
    .
    .
  )
)

```

The module diagram for Essential_IO is illustrated in Figure 16 (16:119).

Syntax Checking Expert System Requirements

The IDEF₀ Syntax Expert System should allow the user to check the hierarchical activity IDEF₀ syntax and the boundary IDEF₀ syntax in any diagrams using the facts file created by the retrieve procedures. CLIPS/Ada is interfaced with the essential subsystem and has been proven to be effective. A chain that is searched or traversed from the initial state to the final state of a problem, during which, certain types of solutions are achieved is called a forward chain¹. That means the chaining is reasoning from facts to the conclusions which follow from the facts. It is also known as data driven, bottom-up reasoning(10:159-166).

The primary method of representing knowledge in CLIPS is a rule. A rule is a collection of conditions and the actions to be taken if the conditions are met. The rules were defined to describe how to solve a problem. The entire set of rules in an expert system is called a **knowledge base**. CLIPS provides the search mechanism (the inference engine) which select the facts in the data base to be matched with the condition(s) in the rule base and continue on this cycle until there is no rules eligible to be fired. The current state is represented by a list of facts. Here the facts or the data for the data base is the facts retrieved by the Essential_Fact.Utilities. The rule base is to be applied by the inference engine integrated as CLIPS/Ada to the facts data file. As the LHS of a rule are met, the rule are activated and placed on the agenda according to their priority. The priority is default to 0 for every rule in the knowledge base, unless a salience declaration is placed at the first pattern of the rule to change it. A rule with the highest priority, once it is activated will remain at the top of the agenda, thus will be fired first. After no rules are eligible to be activated, the top rule on the agenda is selected, and its RHS actions are executed. As a result of RHS actions, new rules can be activated or deactivated.

This pattern matching, activation, firing rules cycle is repeated until all rules that can fire have done so or until the rule limit is reached(21:1-5).

¹The forward chain is different from backward chain in which that a backward chain is traversed from a hypothesis back to the facts which support the hypothesis

The expert system to be developed here must not only be able to check the syntactical limitations for each activity, but also be able to find the inconsistencies between hierarchical IDEF₀ diagrams. Thus, provide the user with Error, Warning, Suggestion or Notice messages. A summary of the functions are listed below:

- Check that each activity must have at least one input and output.
- Check that each activity must have a name and be numbered.
- Warning the user that any particular activity has too many data element associated with it, or the activity has some information, for instance, a description of the activity is missing.
- In the hierarchy of the IDEF₀ diagram, each parent activity's boundary data elements must be consistent with its child data elements.
- The number of icom number of a parent diagram and its child diagrams must be the same.
- The icom code of a parent activity should be consistent with its child diagram too.
- The number of boundary input, output, control, mechanism consistency check between a parent and its child activities.
- Utility and Auxiliary rules to build up the environment of the syntax checking file.

Once the syntax checking function in the Essential Model menu is selected, the user should get a list of messages concerning his work. If an error was encountered during the syntax checking, the subsystem will be halted by a particular rule in the rules file. Otherwise, a congratulatory message will follow all the Warning, Suggestion, or Notice messages if there are any.

The CLIPS/Ada used the VAX Ada Compiler version 1.5 running under VAX-VMS 5.1.1 to create the executable. Therefore, it is useless on Unix based machines. Since the primary platform for this research is a SUN-4 running a version of UNIX and a Verdix Ada compiler, several changes to the original source code are performed by (16). First, all the CLPS/Ada source code files had

.ADA or .ADS extensions that are unacceptable to Verdex were changed to .a and .spec.a files. All the files was transferred to Olympus to be integrated as the Essential Subsystem of SAtool II. When compiling CLIPS/Ada, many warning messages are still received. These messages are due to the source code authors explicitly declaring loop counters. VAX Ada obviously allows such declarations; Verdex Ada allows them also but does not particularly care for them. Therefore, Verdex Ada issues a warning message(16:132-133). Those warning messages can be ignored. Also, the objective of this study is to develop a structured expert system to evaluate application facts and rules based upon expertise in the future.

Summary

This chapter presented the requirements analysis for the development of IDEF₀ Diagram Translator and IDEF₀ Syntax Expert System. Since the Essential Subsystem of SAtool II is entirely based on Ada language, the Expert System will be done using CLIPS/Ada. The number of fields of the facts format are unlimited, thus giving us freedom to define the format of our expert system checking rule patterns.

All the facts information of the essential subsystem can be translated into facts format files, one file for the expert system and one for the essential model. Another expert system syntax checking file can be loaded with the facts file in the CLIPS/Ada working memory. As a whole, those files should be able to include all the information provided by the facts file and provide necessary error messages and editing suggestions for the user to save their manual labor of checking the syntax and consistency of the user's IDEF₀ hierarchy diagrams.

IV. HIGH LEVEL DESIGN

Introduction

The purpose of this chapter is to present and justify the preliminary software design for the IDEF₀ Diagram Translator (IDT) and the IDEF₀ Syntax Expert System (ISES). The idea and principles of SADT is followed throughout the design process. The IDT is an object called the "Essential Fact Utility" and is implemented in the Essential Model. The ISES is a CLIPS file containing all the knowledge base of the syntax checking rules. The IDT is a set of Ada procedures to extract the data in Essential Model data structure and put this data into individual data facts files. The emphasis here focuses on the design and implementation of the Syntax Expert System checking rules. There are four stages in expert system development:

1. problem selection
2. initial prototype
3. expanded prototype
4. delivery system (7:23)

The design of the IDT and the expert system are currently developed. But, the facts format resulting from the IDT is the data format of the expert system. So the implementation of the expert system heavily depends upon the implementation of the IDT.

The Essential Model of SAtool II is not complete. The Syntax Expert System will be the initial prototype of the expert system as a subsystem of the SAtool II. The user should be able to create their hierarchical IDEF₀ diagrams, store and restore their file and perform syntax checking functions using The Essential Model.

The underlying efforts for this thesis investigation include the development of the knowledge for understanding the background of SAtool II, the data structure of the Essential Model and the

application of knowledge based systems. Since AI systems do more than process data for the user; they use knowledge to improve their functionality. Expert systems navigate through knowledge bases to solve problems and build new paths around rules and data. Knowledge development, that's the real answer(2:5).

Previous Study Considerations

The Sun3 and the Sun4 workstations using the SunOS and the SunView window-based environment are required for this tool. Also the IDEF₀ validation tool is implemented with Ada in order to translate the essential model IDEF₀ diagrams into CLIPS/Ada readable facts format. It is implemented as an Ada object called Essential Fact Utilities.

The expert system syntax checking functions developed in (16) has only validated its feasibility. Much more syntax checking rules are to be implemented, especially for the consistencies of *boundary arrows* between a parent and its child diagrams. Once the two main objects are completed, they will be integrated into the Essential Model together with the CLIPS/Ada performing the syntax checking functions in SAtool II.

IDEF₀ Diagram Translator

The translator is used to translate the IDEF₀ graphical features extracted from the Essential Model Object managers into a set of facts formatted for output to a file for permanent storage or for input to the CLIPS/Ada working memory for syntax checking. It is required to be implemented in Ada language. Ada is a strongly typed, high level language based on a set of easily understood concepts, such as data abstraction, information hiding, and strong typing. In a sense, Ada is a language that directly embodies many modern software engineering constructs and is therefore an excellent vehicle with which to express programming solutions (4.4). The Flow diagram for the IDEF₀ diagram translator is illustrated in Figure 17.

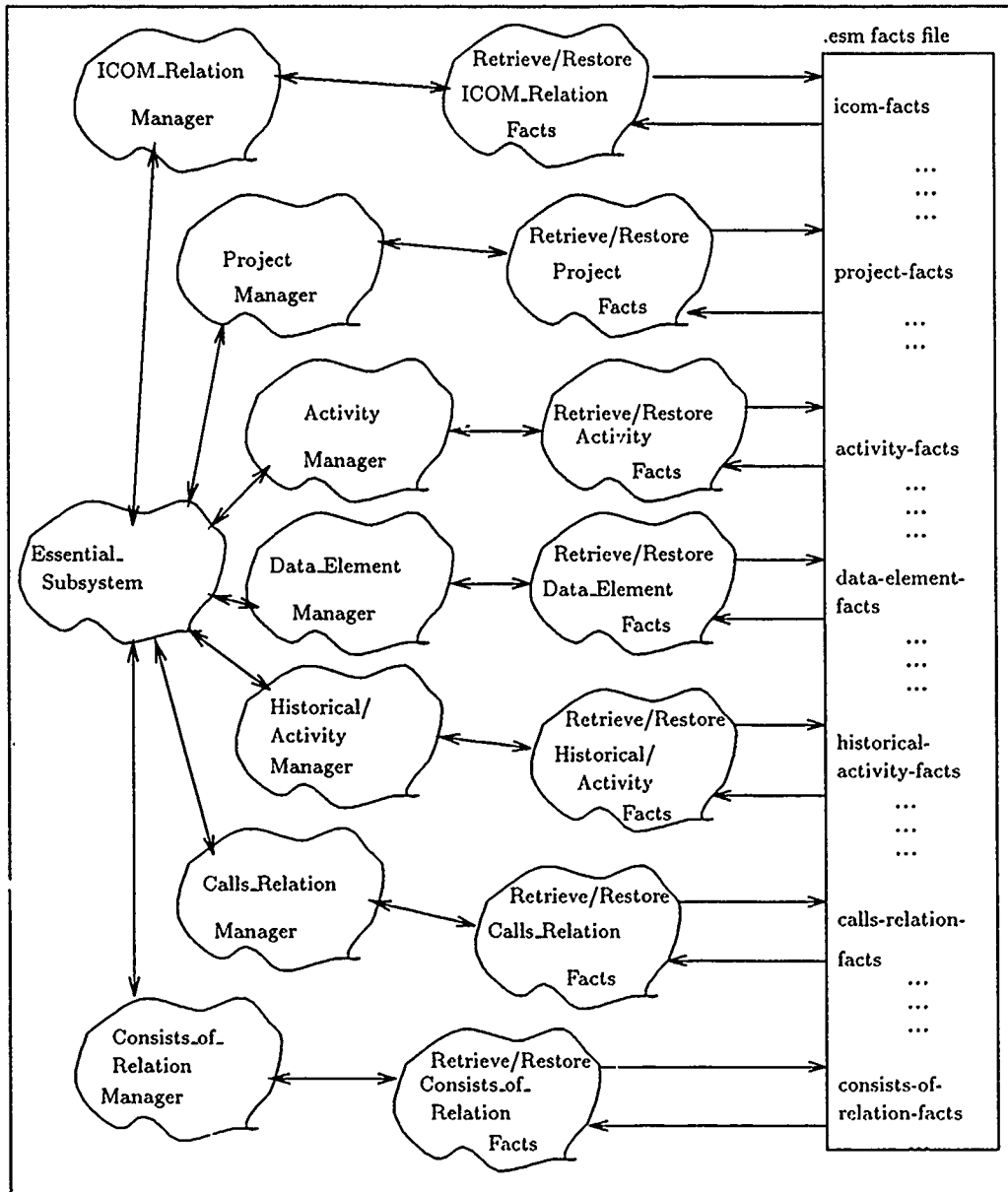


Figure 17. Flow Diagram for IDEF₀ Diagram Translator

Because there are seven Objects Classes and Attributes Based on the Essential Data Model. So there are seven sets of Retrieve and Restore procedures for each of those seven object classes.

- ICOM Relation Manager
- Project Manager
- Activity Manager
- Data Element Manager
- Historical Activity Manager
- Calls Relation Manager
- Consists Of Relation Manager(16)

All those procedures are within a package named *Essential_Fact_Uilities*. Through interfacing with the object *Essential_IO* in the data model, each set of the facts extracted from the managers are given a name by the statement “(deffacts the-name-of-the-facts (fact-1) (fact-2)...),” in the package *Essential_IO*. Where the ‘deffacts’ is a CLIPS construct for naming a facts file. Thus the facts extracted from the *ICOM_Relation_Manager* will output a file name *icom-relation-facts* following a set of facts extracted from the manager. The seven manager names and their facts names stated by the *Essential_IO* in addition to their facts attributes is listed in Table 1. This format is initiated by (16) and completed in this investigation. Notice: those fields in parenthesis with capital letters means a fact variable to be extracted from the managers. To understand the meaning of those attributes and the value of fact’s variables should refer to (16).

If any variable in the Essential Model is empty, than the *Fact_Utility* will input a “null” string into the facts format. If the fact to be extracted is not empty and multi-field, like activity descriptions, than the file for Essential Model will save all the lines of the description and the file for the CLIPS working memory will only save a ‘not-null’ for the syntax checking expert system. The expert system needs only to know that the description is not null. Remember that the facts

format created by the Essential_Fact_Utility creates all the facts input by the user for the IDEF₀ diagrams. It does not show the boundary arrow data element relations between any parent and its child activities.

An arrow in the IDEF₀ diagram may connect with functions on the drawing at both ends is called an *intermediate arrows*. If one of the ends may be unconnected, it represents a *boundary arrow*. Boundary arrows indicate that the information is produced or consumed beyond the scope of the particular drawing. Boundary arrows at the A-0 level are referred to as *external arrows* which represent constraints of the external environment and outputs to that environment. An important aspect of maintaining completeness and consistency in an IDEF₀ model is to make certain that all such boundary arrows match between a box and its lower level decomposition. As listed in Table 1, the ICOM codes are represented as 'i' for input, 'c' for control, 'o' for output and 'm' for mechanism which represents the ICOM relation of the data element arrows to the activity. They must be based on the *relative positions of the arrows on their parent diagram* where they meet the edge of the parent box. Thus a particular boundary arrow of a child diagram should have the same ICOM code as their parent diagram. Furthermore, a *tunneled arrow* represents a discontinuity that a constraint may arise that was not shown on the parent function or a constraint may not be appropriate at lower levels of detail. A new constraint that was not presented on the higher level diagram is shown as a boundary arrow with parentheses "()" around its unconnected end. Any constraint that is not represented in a lower level decomposition is indicated with parentheses where the arrow attaches to the appropriate box.

For the intermediate arrows, there are two special representations:

1. *feedback* occurs when the output of each function provides an input constraint to the other.
2. *iteration* occurs when the output of each function provides a control constraint to the other(20:13-30).

But the two special representations are not within the scope of this thesis research, since those are not implemented in the Essential_Model. The focus here is concentrated on the implementation of the IDEF₀ diagram translator for the data elements that can be created in the Essential_Subsystem.

The boundary arrow relationship between an IDEF₀ parent diagram and its child diagrams will be created by the *Expert System Syntax Checking* rules before actual hierarchical syntax checking took place. More details are discussed later in Chapter 6.

Since the Essential Model developed in (16) was following an Object Oriented Design and Implementation technique, the Essential_Fact_Utility is implemented as an object in the Essential Model, as defined in (25:14-15). An object is an abstraction of a set of real-world things such that:

- all of the real-world things in the set—the instances—have the same characteristics.
- all instances are subject to and conform to the same rules.

The facts format to be created is a set of real-world things to be manipulated by the Syntax Expert Checking Rules thus a series of “*expert advises*” will be derived for the user of the tool.

Retrieve Procedures. Because all the data of the IDEF₀ diagrams created by the SAtool II user is stored as an Ada record in the Essential Model, the Retrieve Procedures are a set of operations which iterate through all the data structures of the Essential Model. The data structures data records will be Extracted by the retrieve procedures and put those data records into a specified facts format in which each column is strictly defined according to the data element data type in the Environment Types of the Essential Subsystem. The features of Ada language was specified in the book “*Ada as a second language*” (8).

Each object class in the Essential Model will have a set of facts extracted from the data structure and a given facts name by (defacts) as illustrated in Table 1. The facts format is readable to the CLIPS/Ada syntax checking rules. Thus, all the structures and data elements of

Table 1. Object Classes Managers and Facts Format Extracted by Essential_Fact_Uutilities

Object Class/Manager	facts format created
ICOM Relation	(defacts icom-facts (icom-tuple Activity Data_Element ICOM Pair_Id) (icom-activity-inputs Activity_Name #) (icom-activity-control Activity_Name #) (icom-activity-output Activity_Name #) (icom-activity-mechanisms Activity_Name #))
Project	(defacts project-facts (project-name Project_Name))
Activity	(defacts activity-facts (act-name Name) (act-numb Name Number) (act-desc Name Description) (act-has-child Name Child) (act-ref-type Name Reference_Type) (act-ref Name Reference) (act-version Name Activity_Version) (act-ver-chg Name New_Version) (act-date Name Date) (act-author Name Author))
Data Element	(defacts data-element-facts (data-element-name Name) (data-element-type Name Data_Type) (data-element-minimum Name Minimum) (data-element-maximum Name Maximum) (data-element-data-range Name Data_Range) (data-element-values Name Values) (data-desc Name Description) (data-ref Name Reference) (data-ref-type Name Reference_Type) (data-ele-ver Name Version) (data-e-v-chg Name Version_Change) (data-ele-date Name Date) (data-ele-author Name Author))
Historical Activity	(defacts historical-activity-facts (historical-tuple Project Activity_Number))
Calls Relation	(defacts calls-relation-facts (calls-relation-tuple Activity Project Activity_Number))
Consists Of Relation	(defacts consists-of-relation-facts (consists-of-name ID Parent Child))

the users IDEF₀ diagram should be included in the facts file for syntax checking. Even if the user does not input any data for the IDEF₀ diagrams, the procedures should give a 'null' string at the appropriate position in the facts format. An example of its actual output stored for the Essential Model but with only the project name, one activity, one data element is on the following page. Notice its relation and difference with Table 1. In which, Table 1 is the requirements of the facts format of the Essential_Fact_Utility that should be translated from the seven Object Class Managers. The name of each set of facts is named by the Essential_IO with a (defacts facts-name (fact) (fact) ...) statement. While the actual translated output was implemented with all the facts in between a header and an ending of the facts file.

Restore Procedures. In contrast with the Retrieve procedures, the Restore procedures are only those operations that iterate through the facts file and put all the facts back into the Essential Model, the format to restore each piece of fact must be exactly the same as they were as defined in the previous Retrieve procedures, otherwise, an exception is raised and the program stops execution.

IDEF₀ Syntax Expert System Components

The Inference Engine Selected. The inference engine of shell selected for this thesis research is CLIPS/Ada. It is an Ada version of CLIPS, which stands for "C Language Integrated Production System". The selection of shell for the development of any particular expert system has always been a kind of question. "Not a single existing shell will satisfy all the necessities of the developers needs," (7:21-25).

In the technical literature and common usage, expert system shells can lie anywhere on a continuum from interpreters of relatively simple languages to very elaborate development environments. Each has its own purposes and strengths and can complement other shells by being used at different times in a project's life cycle.

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; SAtool II - IDEF0 Essential Fact File - CLIPS Readable Format
;; Date and Time of File Creation : 02/25/91 22:24:11
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;**START ALL FACTS**
(deffacts icom-facts
(icom-tuple Format_Example          Format_Data          c          1)
)
(deffacts project-facts
(project-name Format_Example)
)
(deffacts activity-facts
(act-name Activity_Name)
(act-numb Activity_Name          null)
(act-desc Activity_Name          null)
(act-has-child Activity_Name      null)
(act-ref-type Activity_Name      null)
(act-ref Activity_Name           null)
(act-version Activity_Name        null)
(act-ver-chg Activity_Name        null)
(act-date Activity_Name           null)
(act-author Activity_Name         null)
)
(deffacts data-element-facts
(data-element-name data_format)
(data-element-type data_format          null)
(data-element-minimum data_format        null)
(data-element-maximum data_format        null)
(data-element-data-range data_format     null)
(data-element-values data_format         null)
(data-desc data_format                  null)
(data-ref data_format                   null)
(data-ref-type data_format              null)
(data-ele-ver data_format                null)
(data-e-v-chg data_format                null)
(data-ele-date data_format               null)
(data-ele-author data_format             null)
)
(deffacts historical-activity-facts
(historical-tuple Format_Example          A0)
)
(deffacts calls-relation-facts
(calls-relation-tuple Call_Activity          Format_Example          A0)
)
(deffacts consists-of-relation-facts
(consists-of-name      1 Format_Data          formatted_data)
)
;;**END ALL FACTS**

```

All the shells have four features in common:

- 1. the minimum feature set of a knowledge representation scheme*
- 2. an inference or search mechanism*
- 3. a means of describing a problem*
- 4. a way to determine the status of a problem while it is being solved(7:21-22)*

Here, in this research, the problem to be solved is represented in a set of facts lists translated from the Essential Model Data Structure. Each fact has limited number of fields. The knowledge base is another file of rules that will be activated by the inference engine, examining, featuring, and changing the status of the problem until there is no rule eligible to be applied. Thus, a set of certain results is derived through the process and expert suggestions is introduced to the user.

Knowledge Base. Knowledge base is the heart of an expert system. It contains the problem-solving knowledge of the particular application. CLIPS was selected as the shell tool for this thesis research. The designer of an expert system should have a full understanding of both all the application techniques of a knowledge base (21), and all the details in the problem domain. Thus, the knowledge base will be able to reflect all the necessary characteristics intended. In the development of an expert system, all the knowledge bases implemented are in the form of if ...then rules. A rule is a collection of conditions and the actions to be taken if the conditions are met. The developer of an expert system defines the rules which describe how to solve a problem. The entire set of rules in an expert system is called a knowledge base. Some good examples are illustrated in (22) about CLIPS rule developing guides.

The knowledge base here is required to check the IDEF₀ syntax features like: each activity should have a name, number, description, each activity box should have at least one control and one output arrow; the parent activity boundary in arrows should be consistent with their child activities boundary arrows, etc.

The knowledge base must be able to derive the relationship between a parent diagram and its child diagrams. It cannot check all the required features directly from the facts created by the Essential_Fact_Utility. Hierarchical rules in the knowledge base to build up boundary relations between any particular parent and its child diagrams through the fact created are necessary.

If any syntax inconsistencies were found by the knowledge base, an appropriate message should be provided to the user of the condition detected. through which, the user could easily go back to correct the errors in his file without time consuming and error pruning manual checks.

As the IDEF₀ syntax does suggest that any parent activity should not have more than six child activities, the rules to be developed here should consider those parent activities with two, three, four, five and six child activities. But a set of rules should inform the user that any any particular activity has more than six child activities.

Data Base (Working Memory). The data base contains a vroad range of information about the current status of the problem being solved. The temporary output files of the IDEF₀ Diagram Translator became the initial data base for the Syntax Checking Expert System knowledge base. A package named CLIPS Working Memory Interface in the Essential Model is the only object that has direct interface with the CLIPS/Ada. All the related files must through this interface to accomplish the Expert System Syntax Checking functions(16:86).

While checking the IDEF₀ syntax, the data base also contains a list of rules that have been examined and fired. The contents of the data base is volatile, the changing of its contents may very well affect the execution of the knowledge base.

User Interface. The user interface allows the user to communicate with the system and also provides the user with an insight into the problem-solving process carried out by the inference engine. The user interface adopted here is the menu selection in the Essential Model. The advantages of using a menu-based interface are as follows:

1. Users need not know the names of individual commands.
2. Typing effort is usually minimal.
3. It is impossible for users to put the system into an erroneous state.
4. Context-dependent help can be provided(26:265).

An example of the program test and demonstration through the menu selection user interface is in Appendix D. The input file "thesis_err.esm" is an output facts file of the Essential_Model, it is used to be restored back to the Essential_Model to check its IDEF₀ syntax. It was specially designed to project the syntax checking abilities of the expert system. The resulted syntax checking error messages are all commented with the origin of their errors.

Test Plan

A bottom-up testing methodology is used because IDEF₀ Diagram Translator and IDEF₀ Syntax Expert System are lower-level than Satool II. The testing steps are : unit testing, integration testing, and validation testing(26:502). The Unit testing step focuses on each module individually to make sure that its functions properly as a unit. Thus the IDT should have all its procedures correctly executed and the facts file extracted from the Essential Model should be exactly the format as defined.

For the level of syntax checking rules, each group of rules is individually tested to make sure that the behavior of its execution is under control and desired results will be created. Also an example project of hierarchical IDEF₀ diagrams provided in Chapter 2 named "Control Elevator", will be used for validation testing on the over all functions of the system. Carefully designed errors, including parent activities with 2, 3, 4, 5, and 6 child activities are expected to be detected by the Expert System Rules. The same set of IDEF₀ diagrams but with designed error inputs is also presented in contrast with the sample IDEF₀ diagrams. Those errors are analyzed and explained with added comments in Appendix D: SAMPLE ESSENTIAL MODEL IDEF₀ SYNTAX CHECKING SCRIPT. Each

syntax checking messages will be justified to prove that the system is functioning as it is designed to be. Thus we will be confident that we are building the right product.

The total number of rules for syntax checking expert system is 198, not including 43 auxiliary rules. A subset of the names of the rules in the rule base is listed below, it is listed as an example for the overview of the rules been implemented. The complete file of rules and their implementation is in Appendix C.

1. print-introduction
2. print-project-name
3. exit-if-error
4. no-error-congratulate
5. zero-outputs
6. zero-controls
7. too-many-mechs
8. too-many-outputs
9. too-many-controls
10. too-many-inputs
11. null-project-name
12. null-activity-number
13. null-activity-description
14. too-many-children-level1
15. too-many-children-level2
16. too-many-children-level3
17. parent-2child
18. parent2-boundary
19. child2-boundary-child1
20. child2-boundary-child2
21. clear-2child-mid
22. remove-2child-2boundary
23. rid-2child-2consists
24. check-2child-parent
25. check-2child-parent-consists
26. check-2child-parent-icom

27. check-2child-child
28. parent2-icom-c
29. parent2-icom-o
30. parent2-icom-i
31. parent2-icom-m
32. parent2-control-add
33. parent2-output-add
34. parent2-input-add
35. parent2-mech-add
36. child2-icom-c
37. child2-icom-o
38. child2-icom-i
39. child2-icom-m
40. child2-control-add
41. child2-output-add
42. child2-input-add
43. child2-mech-add
44. check-parent-2child-control
45. check-parent-2child-output
46. check-parent-2child-input
47. check-parent-2child-mech

Summary

This chapter presented a high level software design for the IDEF₀ Diagram Translator and the IDEF₀ Syntax Expert System. The facts format to be created by the translator and to be checked by the expert system are described. The concept of an Expert System was explained and the knowledge base to be implemented for the expert system in this research was analyzed both on its functional basis and on its structure.

The preliminary test design expectations were introduced. These provide a guide to the low level design in the next chapter. A list of the name of a subset of those rules is summarized as follows: The rules name listed here only shows a parent activity having two child activities. For those parent activities with three to six child activities, the rules name are not shown here, but

their names are similar, except the changing of the number in those rule names indicate that this rule is for a parent activity with that number of child activities. Also, more intermediate rules are needed for those rules. The increasing of child number increases the complexity in implementing those rules.

V. DETAILED DESIGN, IMPLEMENTATION, AND TESTING

Introduction

This chapter discusses the low level design and implementation of the IDEF₀ Diagram Translator and the IDEF₀ Syntax Expert System specified in the previous chapter. As mentioned previously, the facts format to be created by the IDT must be correct before further effort is expended to implement the syntax checking rules for the Expert System. Those facts are the initial data base (working memory/knowledge base) for the IDEF₀ Syntax Checking Expert System.

The construct of syntax checking rules has been discussed in chapter 4. The rationale and detailed implementation of those rules is explained in this chapter.

IDEF₀ Diagram Translator Implementation

The IDEF₀ Diagram Translator is implemented as an Ada package named `Essential_Fact_Utility`. It has seven pair of Retrieve and Restore procedures. Since the procedures for ICOM relations and Project name are already completed in (16), the remaining work will have to complete the following procedures:

1. Activity:
 - Retrieve Activity Facts
 - Restore Activity Facts
2. Data Element:
 - Retrieve Data Element Facts
 - Restore Data Element Facts
3. Historical Activity:
 - Retrieve Historical Activity Facts
 - Restore Historical Activity Facts
4. Calls Relation:

- Retrieve Calls Relation Facts
- Restore Calls Relation Facts

5. Consists Of Relation:

- Retrieve Consists Of Relation Facts
- Restore Consists Of Relation Facts

The facts file created by this package will carry a '.esm' extension. Its format has already shown in Table 1. The file Essential_Fact_Utility is presented in Appendix B.

Its relations and visibility with the other Ada objects in the Essential Model in addition to the syntax checking rules file is illustrated in Figure 18.

Expert System Syntax Checking Rules Design

The process to develop a rule based expert system has many steps:

- planning
- scheduling
- chronicling
- analysis
- configuration management
- resource management

First the feasibility of this approach is demonstrated in (16). A design goal was set to implement IDEF₀ syntax checking expert system for SAtool II. The facts translated to represent IDEF₀ diagrams consists of one or more fields enclosed in matching left and right parentheses. Refer to Table 1.

The relative position of each field in a fact translated by IDEF₀ Translator is strictly defined. The space between each field might be different but they will be neglected by CLIPS if the spaces are

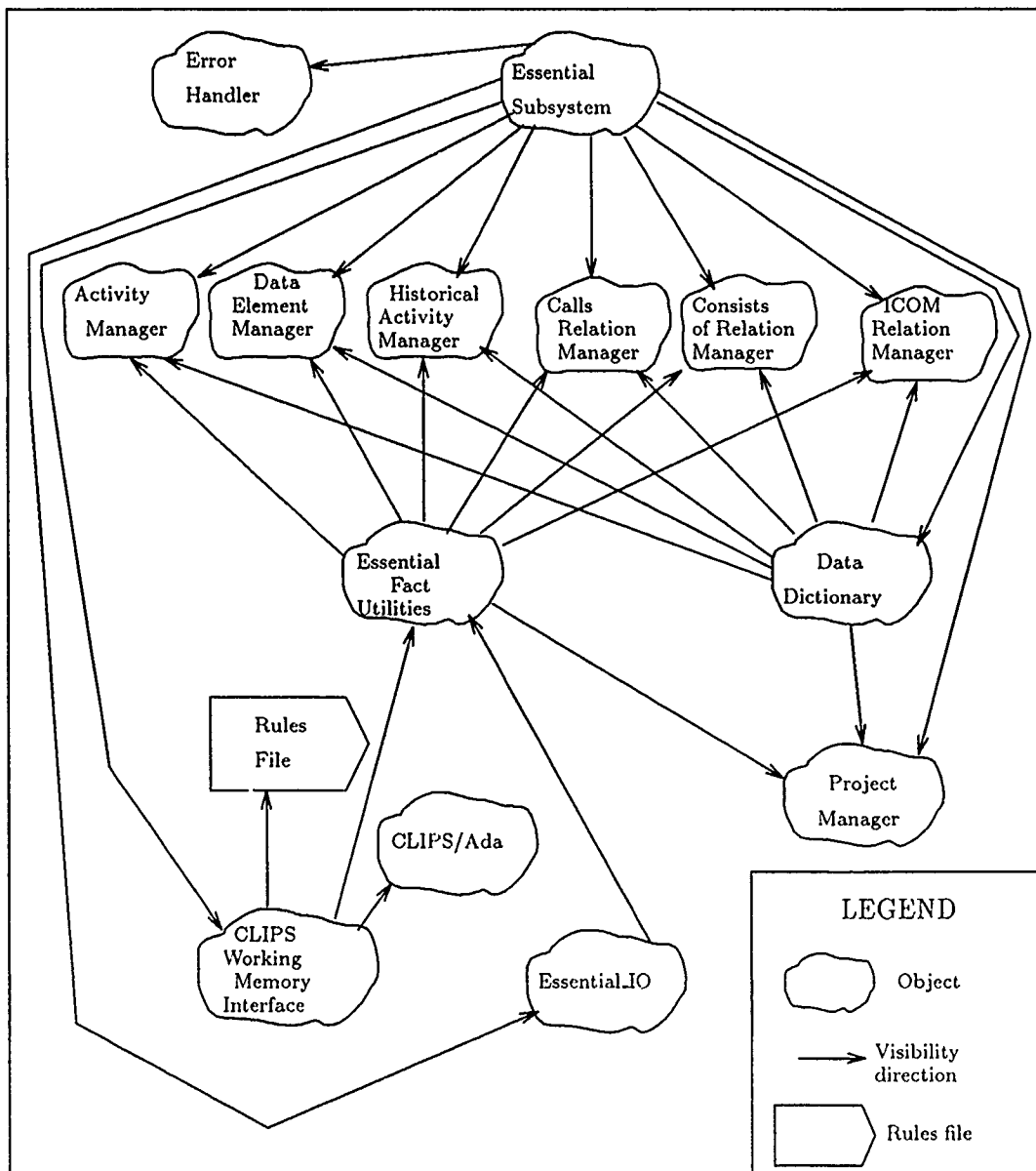


Figure 18. Essential Subsystems Relations and Visibility

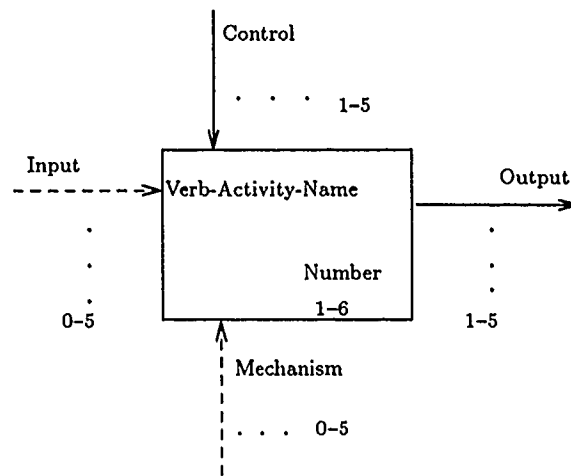


Figure 19. A Typical Activity Box Features

more than one. Some of the IDEF0 Diagram syntax can be directly derived from the facts created by the IDT, but in practice most of them cannot. The design process evolves as intermediate rules are implemented to create the data facts needed to check particular IDEF₀ syntax. For instance, the number of boundary arrows between a parent activity and its child activities. To be successful, the implementing techniques of CLIPS must be developed through out this effort. More efficient rule sets are gained from the experience of the previous rules implemented. Thus structured and related rule bases are expected to be developed in order to capture all the syntactical features of IDEF₀ diagrams.

IDEF₀ Diagram Syntax Analysis. Since the IDEF₀ system model consists of a series of hierarchically related function diagrams, each function box has to have some required syntax features. Also the relation between a parent box and its child box must be consistent with each other.

Activity IDEF₀ Syntax. A typical activity box is shown is Figure 19. If any necessary features for a box is missing, then its syntax is incorrect.

The IDEF₀ syntax for an activity box is:

- An activity box must have a name started with a verb.
- An activity box must have a number except the top-most level A-0 diagram.
- An activity box must have at least one control, one output but no more than five.
- An activity box may have zero to five input or mechanism.
- Except for the top-most level Context Diagram, there should no more than six boxes in a diagram.
- Any arrows or data connected to the box should be named.

Boundary IDEF₀ Syntax. As mentioned in Chapter 4, the relative positions of a boundary arrow of child activities must meet the edge of its parent activity. The boundary IDEF₀ syntax for a parent activity and its child activities are listed below:

- A parent activity must have at least two but no more than six child activities.
- The total number of input, output, control, or mechanism arrow(s) of a parent activity must be the same as those of its child activities boundary input, output, control, or mechanism arrow(s).
- Each boundary parent or child arrows must have a data name.
- The data name and icom relation of each boundary arrow between the parent and its child diagrams should be consistent.
- Any boundary control and output numbers should not be less than one and more than five. Unless a pipeline data item is used at the boundary.
- Any boundary input and mechanism numbers should not be more than five, but might be 0

At this point, we must remember that the **intermediate arrows** between activities will be the boundary arrows for each individual activity. And should be the next lower level boundary arrows of the child activities for that particular activity. Notice the **mid** data element in Figure 20.

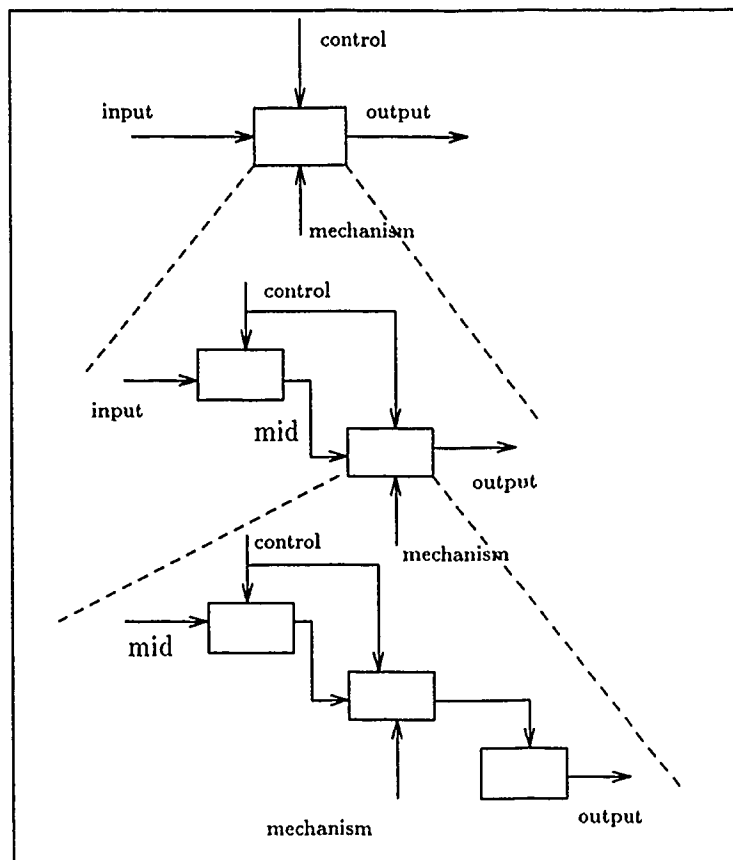


Figure 20. Hierarchical Boundary Relations Between Parent and Child Activities

Syntax Checking Environment. Since the (initial-fact) for any Working Memory always exists; the title message of the syntax checking environment is created by using this fact and with a highest salience declaration to ensure that the environment will be created before any other messages. Right after this, the project name will be directly derived from the facts and presented after the syntax checking environment message. It reads as follows:

**** Essential Subsystem Syntax Checking Messages ****

⇒ The project == Name-of-Project == is being checked:

After all the checking rules were fired, if no errors were discovered, than a congratulatory message will be presented, but a suggestion will also be presented to remind the user recheck the logical structure of his work. Otherwise, when syntax error occurred, another rule with the lowest salience will be fired to halt the program preventing further rules firing. The control is returned to the top-level program. This rule must be the last one to be fired, because we want to make sure that all applicable checkings are all fired. Thus all information available to the user should be presented before the program halted.

Essential Model Facts Format Analysis for Boundary Arrows. Since all the data elements (arrows) related to an activity are only represented in the icom facts.

(icom-tuple Activity Data_Element ICOM Pair_Id)

And the parent child relations between an activity and its child activities are represented in the activity facts.

(act-has-child Parent_Activity Child1_Activity)

(act-has-child Parent_Activity Child2_Activity)

There is no direct trace of the boundary arrows of a particular parent activity and the boundary arrows of its child activities. Thus hierarchical levels of rules must be developed before actual syntax checking can be performed. But there are still some features of the IDEF₀ syntax that could be directly derived from the facts created by the IDT.

For instance, each activity box must have at least one control and one output; each activity must have activity number, descriptions and the project must have a project name. Those can be directly derived by the facts created by Retrieve and Restore ICOM Relation procedures in the IDT:

(icom-activity-control Activity_Name #)

and

(icom-activity-output Activity_Name #)
(act-name Name)
(act-numb Name Number)
(act-desc Name Description)
(project-name Project_Name)

If any of those field are missing, then the IDT will put a 'null' in the appropriate field, thus the checking of these missing fields are easier to implement. Say for activity description, if it is null than the fact should be:

(act-desc Activity_Name null)

If this fact pattern is matched by the LHS of a rule named "null-activity-description" with only this pattern, and the 3rd field in act-desc fact is a 'null', then the RHS action could be fired :

(defrule null-activity-description

```
(act-desc ?activity-name null)
=>
WARNING: ?activity-name has no description.
```

It must be mentioned that this is only an example to show the matching of a pattern in the data base and a pattern in the LHS of a rule. The CLIPS syntax for defining a rule is not strictly followed here. Refer to Appendix C for the detail implementation of Syntax Checking Rules.

Translation Rules for Boundary Arrows. Since the boundary arrows cannot be directly derived from the facts created by IDT, levels of rules are necessary for creating those facts between each parent and their child diagrams. Recall the constraint that each parent should have at least two but no more than six child diagrams. For each level of rules, there are five group of rules for any parent activity with 2, 3, 4, 5, or 6 child activities. The relations between each parent and its child activities are distinguished by the parent name derived from the facts:

```
(act-has-child Parent_Name Child1)
(act-has-child Parent_Name Child2)
```

are in the original state of the Essential Model showing a parent child relation. Since in this example, the parent activity, Parent_Name, has only two child activities. A new parent/child relation fact should be created as:

```
(parent2 Parent_Name Child1 Child2)
```

This format of fact should be created for any parent activity with two child activities at any level of the IDEF₀ diagrams with different Parent_Name and child names.

Different parent and child boundary relations should also be created for syntax checking.

Those boundary facts might be created at different time and stored in different places in the new fact lists created and asserted in the Working Memory.

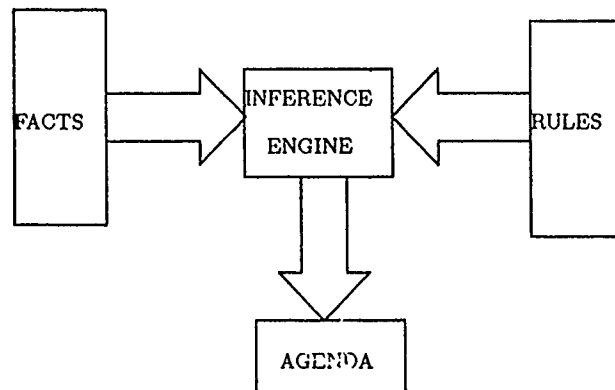


Figure 21. Pattern Matching: Rules and Facts

In rule-based languages, however, the matching process takes place repeatedly. Normally, the fact-list will be modified during each cycle of execution. New facts may be added to the fact-list or old facts may be removed from the fact-list. These changes may cause previously unsatisfied patterns to be satisfied or previously satisfied patterns to become unsatisfied. The problem of matching now becomes an ongoing process. During each cycle, as facts are added and removed, the set of rules that are satisfied must be maintained and updated.

It is the rules that remain static and the facts that change. Thus, the facts should find the rules(10:502-503).

As new facts are created, they might add new rules eligible to fire in the **agenda**, which is a stack of rules eligible to fire. On the contrary, as facts are retracted from the facts list, the rules to be fired in the agenda relating to those facts will also be retracted. See the Pattern Matching relation of Rules and Facts in Figure 21.

High Level Creating Boundary Facts Rules. For the designing of hierarchical rules, care must be taken to make sure that the execution of those rules are controlled. Groups of rules are implemented. Thus some techniques or principles must be carefully followed:

1. All the variable names for each group of rules must be distinct and easy to recognize

2. The ordering of patterns on the LHS of a rule should be carefully designed in accordance with the facts sequence in the facts created by IDT to minimize change of states in order to improve efficiency:
 - most specific pattern go first
 - patterns matching volatile facts go last
 - patterns matching the fewest facts go first
3. Perform tests as soon as possible; which means any test patterns within a rule should be placed as close to the top of the rule as possible.
4. Use a priority declaration pattern in a rule to aid in controlling the flow of execution.
5. Use simple rules vs complex rules; the key is to prevent the unnecessary comparisons from occurring.
6. Reduce comparison by using temporary facts to store data(10:502-529).

To create the boundary facts relations between any parent activity and its child activities, the parent-child relation must first be created and stored in a single fact. This is accomplished by a set of rules that create a set of facts each containing the name of a parent activity and its child lists as in the example below:

```
(parent2 Parent-two-child  Child1 Child2)
(parent2 Parent-of-two      Child-1 Child-2)
(parent3 Parent-three-child Child1 Child2 Child3)
(parent4 Parent-four-child  Child1 Child2 Child3 Child4)
(parent5 Parent-five-child  Child1 Child2 Child3 Child4 Child5)
(parent6 Parent-six-child   Child1 Child2 Child3 Child4 Child5 Child6)
```

With those facts created for each set of parent and child activities, the facts to create their boundary relations could then be fired. Adding the pattern matching for icom-tuple, each activity's name, data, icom relations might be created. Using the **parent name** as a key to keep track of any particular parent-child relations, further data facts could be matched between each pair of parent and child activities according to these facts.

For the child activities, it is a little more complex. Since the intermediate data arrows are not boundary arrows. They must be cleared to be consistent with their parent activity. Many types of intermediate arrows are to be considered for parents with 2, 3, 4, 5, and 6 child activities. Those type of intermediate arrows are to be retracted from the *created child boundary* facts. Part of those types are illustrated in Figure 22 as an example. The same or similar situations may be extended to those different parents with different number of child activities.

For those intermediate data arrows that are the parent or child data of a pipeline data item, no matter how many levels of pipeline data items may exist. Those pipeline data relations will be stored in the Essential Model as consists-of-relation facts. Showing that a parent data is having at least two child data, the intermediate pipeline arrows should also be retracted from the boundary fact lists. Two types of consists of relation data arrow forms which is shown in Figure 23.

Low Level Sifting/Adding Rules. **Salience** is suggested not to be excessively employed when patterns can be used to express the criteria for selection(10). Each level of rules must not fire until its higher level rules have already created all the available facts needed to be matched for the lower level of rules. Salience is explicitly used to control the sequence of rules execution. Which means only after all the original boundary facts have already been created, then all the rules used to eliminate intermediate arrows will be fired accordingly. Thus the intermediate arrows will be sifted out of the child boundary arrow facts.

As an example, two different parent but with the same number of three child diagrams should

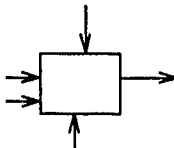
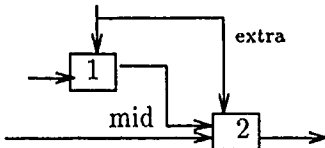
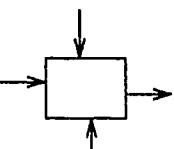
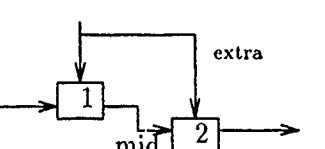
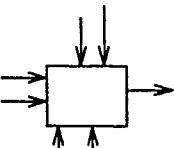
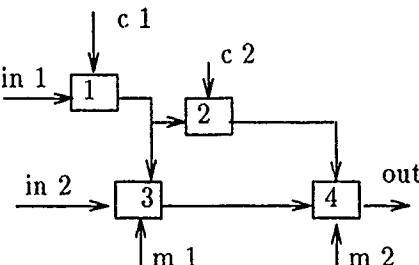
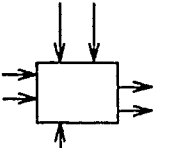
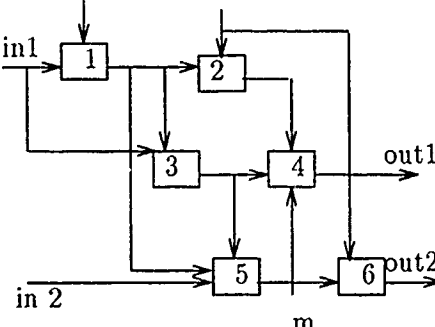
Parent	Child	Analysis
		The out put of child1, input of child2, mid is not a boundary arrow.
		The extra control is only part of control arrows, it is considered only one boundary control arrow.
		only those labeled arrows are boundary child arrows.
		Only those labeled arrows are boundary arrows for the parent activity with 6 child activities.

Figure 22. Intermediate Data Arrows Between Child Activities

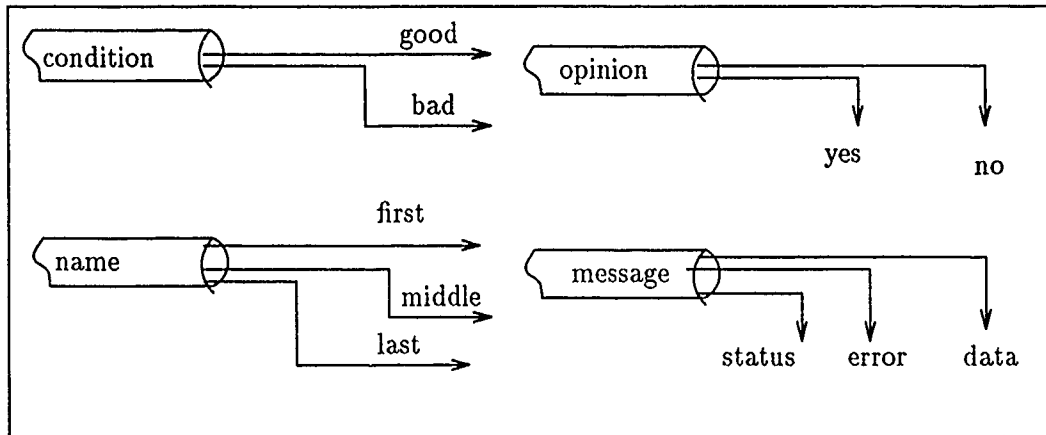


Figure 23. Pipeline Consists of Data Arrow Relations

have their boundary arrow facts created as follows; the key to keep track of any parent and its child relations is the **parent-name** in the second fields:

```

(parent3-boundary parent-a data-one c)
(parent3-boundary parent-a data-two o)
(parent3-boundary parent-a data-three i)
(parent3-boundary parent-a data-four m)

(child3-boundary parent-a child-1 data-one c)
(child3-boundary parent-a child-2 data-two o)
(child3-boundary parent-a child-1 data-three i)
(child3-boundary parent-a child-3 data-four m)
and;
(parent3-boundary parent-b data-1 c)
(parent3-boundary parent-b data-2 o)
(parent3-boundary parent-b data-3 i)

(child3-boundary parent-b child-one data-1 c)
(child3-boundary parent-b child-three data-2 o)
(child3-boundary parent-b child-one data-3 i)

```

Up to this point, only after the boundary facts are corrected created, then the rules to create boundary icom numbers could be correctly fired. The first level of icom number rules will create all the parent and child activity facts with the number of 1. For example, a single boundary fact

created as:

```
(parent3-boundary Parent-name P-data c)
(parent3-boundary Parent-name data-P c)
```

Its icom number fact will be created as:

```
(parent3-icom Parent-name P-data control 1)
(parent3-icom Parent-name data-P control 1)
```

The parent activity with name Parent-name may have more than one control. The next lower level rules will match the facts created and add them up to show the correct boundary icom numbers for each parent and its child activities. Thus, the previous two parent3-icom facts will be removed from the facts list, a new facts will be created as:

```
(parent3-icom Parent-name gen1 control 2)
```

Since we need only to keep track of the icom number here, the data associated with each parent or child is not considered here. So they will be replaced by a different symbol created by (gensym)¹ in the proper field to make sure that this field in the newly created facts has no duplicated data element in other facts. The name of the data is replaced by a series of symbols as, gen1, gen2,... as the rule continues on firing. The basic reason here is that all the data elements in the data dictionary should have a different name. Again, the parents name is the key to keep track of the relation between a particular parent and its child activities.

Thus the final icom number facts for a parent may be stored in the facts lists as:

```
(parent3-icom Parent-name P-data control 3)
```

¹gensym is a CLIPS feature that will create different symbols for the matched data item for each firing of the rule

(child3-icom **Parent-name** C-data control 3)

Note the icom number for parent and child facts with the same **parent name** and the same icom code, which is **control** here, should have the same number added. Otherwise it is an IDEF₀ syntax error.

Hierarchical Consistency Checking Rules. Only after all the necessary facts for the boundary arrow facts between parent activities and their child activities are correctly created. Then the consistency checking rules are ready to be used.

The summary of those *if...then* constructs for certain condition and appropriate actions are illustrated in Table 2.

Notice that the absence of a particular fact in the created fact-list is useful for the syntax checking rules. When a parent having a boundary data arrow with labeled data element name, but its child activities does not have the same boundary data arrow, and the boundary is not a pipeline data, then it is a syntax error. When the parent and its child activities having the same boundary data arrow but with a different icom code, it is still a syntax error.

Through the process of IDEF₀ diagram syntax checking, there are 5 types of messages that might be provided to the user by the Syntax Checking Expert System. Those are summarized in Table 3:

The structure and visibility between each set of rules are illustrated in Figure 24.

Table 2. *if...then* Construct for the IDEF₀ Syntax Checking Knowledge Base

<i>if</i>	condition(s)	<i>then</i>	action(s)	Remarks
if	an activity does not have a name	then	prompt the user a syntax error	The name should started with a verb
if	an activity does not have a number	then	prompt the user a syntax error	The number should started with an A
if	an activity does not have any description	then	prompt a warning about the situation	A description is not required by the syntax
if	an activity does not have any control	then	prompt the user a syntax error	at least one control is required for any activity
if	an activity does not have any output	then	prompt the user a syntax error	at least one output is required for any activity
if	an activity has more than five input, output, control or mechanisms	then	prompt the user a syntax error	input and mechanism may be 0-5 control and output must be 1-5
if	a parent activity has a boundary data, but its child diagrams does not have it	then	prompt the user a syntax error	boundary data element inconsistency between the parent and its children
if	a parent's boundary data icom code is different than it's child's boundary data	then	prompt the user a syntax error	boundary icom inconsistency between the parent and its children
if	the number of a parent's boundary input, output, control and mechanism is different than its child's	then	prompt the user a syntax error	number of boundary data inconsistency between the parent and it's children

Table 3. Possible Syntax Expert System Checking Results

<i>number</i>	MESSAGES	<i>Remarks</i>
1	CONGRATULATORY:	No syntax errors was found. If no syntax error facts was asserted after the rules checking is done, then this message will be presented at the end of all the other messages.
2	ERRORS:	Syntax error encountered, syntax error fact will be asserted, program will be halted after all the checkings are done.
3	WARNING:	Some features of the users project work were discovered that might cause problem.
4	NOTICE:	Reminder to the user that something should be carefully done.
5	SUGGESTION:	Suggest the user that further manually recheck might be helpful to find logical errors that cannot be found by the syntax checking rules.

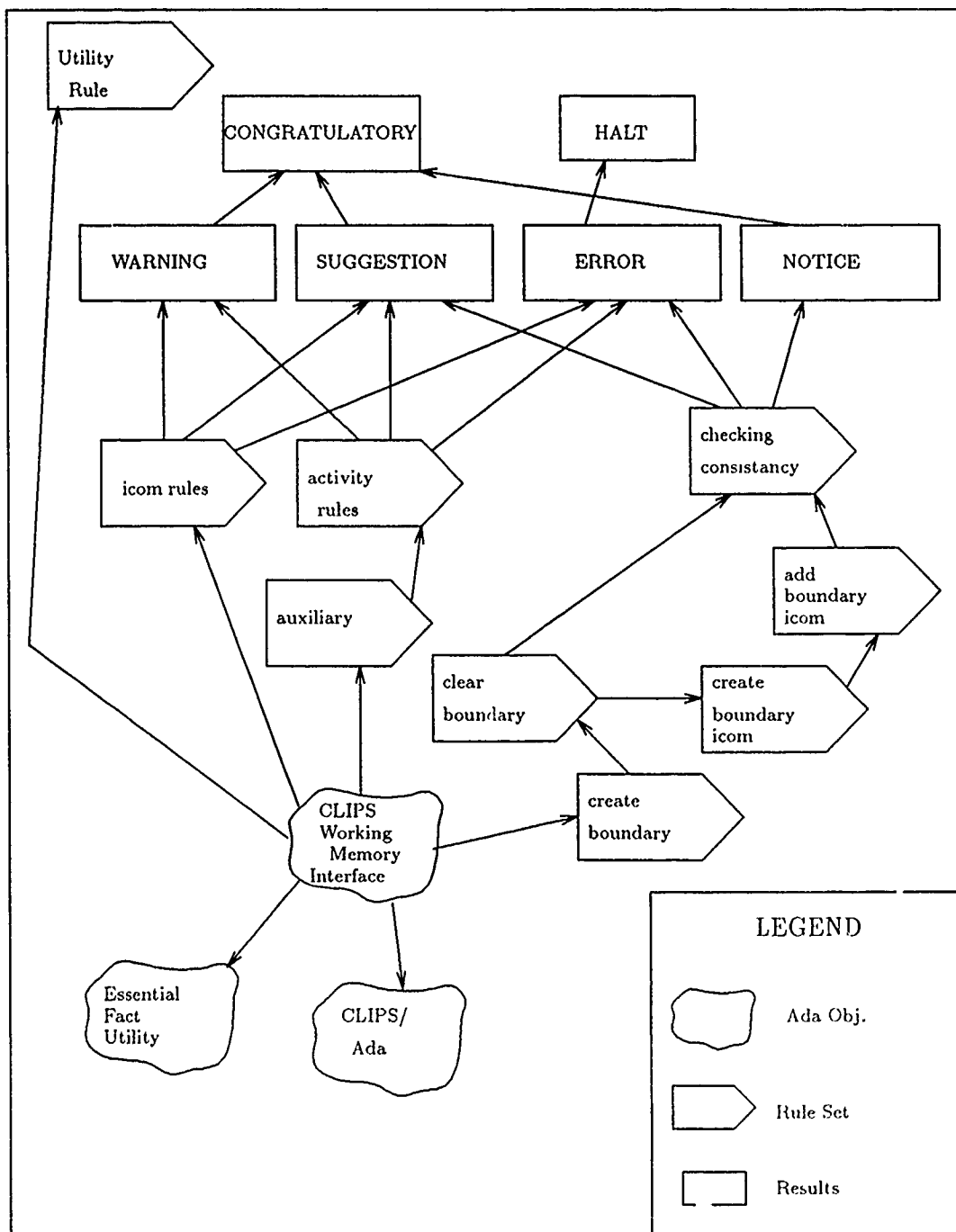


Figure 24. SAtool II Syntax Checking Rules visibility network

The SAtool II Syntax Expert System Syntax Checking Rules are in Appendix C. Total of 241 rules are implemented.

IMPORTANT NOTICE: Because the behavior of CLIPS control cycles, previously activated rules will be fired after those rules activated later than it was activated. If all the rules have the same priority. The agenda for the activated rules is a first come, last out stack operation . Which means first come (rules activated earlier), last out(will be fired later). Considering this, the sequence of rules presented in the thesis is for the clarity to the readers. It is implemented and grouped using another sequence in the Essential Model to gain efficiency in performing the Expert system syntax checking functions.

Testing Expectations

A well designed testing procedures enables the software engineer to derive sets of input conditions that will fully test all functional requirements for a program. This attempts to find errors in the following categories:

- incorrect or missing functions,
- interface errors,
- errors in data structures,
- performance errors, and
- initialization and termination errors (15:4-10).

The program developed should be able to store all the user input IDEF₀ data structures into a set of facts format as shown previously, and the file could be restored back into the Essential Model for further work or changing or permanent store. Only when all the facts are proven to be correct, then the Expert System could be expected to correctly perform its syntax checking abilities.

For the Expert System, a summary of the expected performance for the expert system are as follows:

1. Does it contains all the required syntactical checking rules for the Essential Model?
2. Does it successfully create all the boundary arrow facts for various parent activities with different number of child diagrams to perform further hierarchical consistency checks?
3. Does all the checking rules correctly reflect the syntax errors? Or does some of the rules had been wrongly fired and caused *errors* thus confused the user?

Test Results Validation

As mentioned in the **Test Plan** in Chapter 4, carefully designed *errors* in the example IDEF₀ diagrams named "Control Elevator" was input as the SAtool II Essential Model test program. The output '.esm' file was named as "thesis.err.esm" to be used in the validation test. In which, it includes parent activities that has 2 to 6 child activities. Data inconsistencies was design between each pair of parent and child activities. Also, icom code consistency, activity syntax, and number of boundary icom consistencies together with checking the number of child activities are intended to be reflected by the syntax checking expert system.

The reader of this thesis should refer to the original example IDEF₀ diagrams in chapter 2, compare the **error example** in Appendix D and the syntax checking results to see that the program correctly performs its syntax checking functions.

The results of the syntax checking expert system based on the "error example" is summarized as follows:

- Activity A2 was found has more than 6 child activities.
- Activity A265 Send_Signals has no description.
- Activity Check_Destination needs at least 1 control.

- Data inconsistency between parent activity Sort.Signals data 'o' false_signals and its child diagrams.
- Data inconsistency between child activity Send.Signals data 'o' signals and its parent.
- Data inconsistency between child activity Compare.Signals data 'c' not_confirmed and its parent.
- Data inconsistency between child activity Clear.Destination data 'i' no_stopped and its parent.
- Data inconsistency between parent activity Elevator.Control data 'o' signals and its child diagrams.
- Data inconsistency between parent activity Elevator.Control data 'c' no_floor_sensor and its child diagrams.
- icon inconsistency between activity Elevator.Control and its child diagram Check.Destination.
- Data inconsistency between child activity Sort.Signals data 'o' false_signals and its parent.
- Data inconsistency between child activity Monitor.Arrival data 'c' floor_sensor and its parent.
- Data inconsistency between parent activity Store.Request data 'i' passenger_requests and its child diagrams.
- Data inconsistency between parent activity Control.Elevator data 'c' floor_sensor and its child diagrams.
- Data inconsistency between child activity Elevator.Control data 'c' no_floor_sensor and its parent.
- there might be an ERROR: The number of boundary controls of the parent activity Sort.Signals is 1 control(s) less than its child boundary controls. Are there "consists of" data items at boundary? Please recheck the syntax.
- there might be an ERROR: The number of boundary inputs of the parent activity Manage.Destination is 1 input(s) less than its child bound

- there might be an ERROR: The number of boundary controls of parent activity Elevator.Control is 1 control(s) more than its child activities. Are there “consists of” data items at boundary? Please recheck the syntax.
- there might be an ERROR: The number of boundary inputs of the parent activity Elevator.Control is 1 input(s) less than its child boundary inputs. Are there “consists of” data items at boundary? Please recheck the syntax.
- Data inconsistency between parent activity Manage.Destination data ‘i’ elevator_status and its child diagrams.

All the intended syntax errors were reflected by the syntax checking expert system. In comparison with the “error example” IDEF₀ diagrams one should notice that a data inconsistency between a parent and child activities will raise two error messages. One by the “check_child_parent” rule and one by “check_child_child” rule. The reason for this is twofold, one is to double check the consistencies between each pair of parent and child activities, the other is for the lowest level child activities that has a data arrow inconsistent with its immediate parent activities, the second rule is necessary to check its consistency with its parent activity.

Summary

In this chapter, the low level design and implementation of IDEF₀ diagram Translator and IDEF₀ Syntax Expert Checking rules are explained. The component and levels of design process are illustrated using both figures and tables. The expectations for the testing results together in Appendix D will be examined to indicate validation of this thesis effort.

VI. CONCLUSIONS AND RECOMMENDATIONS

Introduction

The purpose of this thesis investigation is to continue the design and implement an application of expert system for checking the Essential Model SAtool II IDEF₀ syntax and produce an expert system structure for applications using SADT. This chapter presents a conclusion and several recommendations for future researchers.

Conclusions

This investigation is classified into two major categories: The full implementation of IDEF₀ Diagram Translator (IDT) and IDEF₀ Syntax Expert System Rules. The translator for the IDEF₀ diagram is used to translate the IDEF₀ graphical features extracted from the Essential Model Object managers into a set of facts file. The fact file is formatted to output for permanent storage or for input to the CLIPS/Ada working memory for syntax checking. All the facts are represented as a set of parenthesized lists with different number of fields. The IDT was implemented in the Ada language as a package in the Essential Model.

The IDEF₀ Syntax Expert System consists of the inference engine, the knowledge base, the data base, and the user interface. The expert system shell selected was CLIPS/Ada which was already integrated with the Essential Model as a subsystem in (16). The inference engine search process applies the knowledge to the solution of a specific domain using logical reasoning. To check IDEF₀ syntax in any IDEF₀ hierarchical diagrams, the forward chaining control strategy is used. It applies the knowledge base of the problem, manipulates the initial data base, modifies the data, and derives a series of conclusions. The Expert System Rules file is 105 K bytes, and the pattern matching for the syntax checking rules is both memory and time consuming. Even in a Mainframe, where the Essential Model and all the related subsystems are, it takes minutes for the CLIPS/Ada Working Memory to assert all the facts and another minute for the CLIPS/Ada to compile all the

rules. Then the Syntax Expert System could search through all the facts, change states, derive final syntax expert suggestions.

The Expert System Syntax Checking functions produces correct checking results for the IDEF₀ diagrams with only single data items (constraint) used in the IDEF₀ diagrams. Pipeline data item can be very complex. A pipeline may contain several pipelines, and each single pipeline inside it may contain many data items or even pipelines. Furthermore, pipelines could be a *branch*, a *join* or a complex combination of both. It is almost impossible to implement CLIPS/Ada pattern matching relations to check all the combination of levels of those pipeline parent and child or even grand child pipeline data relation facts, eventhough a few typical conditions for pipeline data are considered in this thesis effort. For those errors detected, the expert system reminds the user to check if that kind of error is caused by a pipeline data arrow.

In rule-based languages, the matching process takes place repeatedly. The fact-list is modified during each cycle of execution. During each cycle, as facts are added and removed, the set of rules that are satisfied must be maintained and updated. Having the inference engine check each rule to direct the search for facts after each cycle of execution provides a very simple and straightforward technique for solving this problem. The primary disadvantage of such a technique is that it can be very slow due to the property called **temporal redundancy**. That is, the actions of a rule will only change a few facts in the fact-list. Hence the facts in the expert system change slowly over time. Each cycle of execution may see only a small percentage of facts either added or removed and so only a small percentage of rules are typically affected by the changes in the fact-list. Thus having the rules drive the search for needed facts requires a lot of unnecessary computation since most of the rules are likely to find the same facts in the current cycle as found in the last cycle. Unnecessary recomputation could be avoided by remembering what has already matched from cycle to cycle and computing only the changes necessary for the newly added or removed facts (10.502-504). Unfortunately, it is not a property of the expert system shell, CLIPS/Ada (the Ada version

of CLIPS 4.3). The Ada version of CLIPS 4.3 is undergoing enhancement by *Computer Science Corporation* in Houston, Texas.

All the facts created for a specific hierarchical IDEF₀ diagrams could be rather a large list. It will take both time and memory to perform the associated syntax checking functions. So far, the test program has 753 facts including activities that have 2, 3, 4, 5, and 6 child activities. It is recommended that, once the Essential Model, Drawing Model and the Syntax Checking Expert System of SAtool II has been proven to be applicable, the rules of the expert system are 'fixed'. Then the compiling of those rules should be implemented and stored before the user selects Check Syntax to gain time efficiency. Up to this point, the syntax checking expert system correctly checks the IDEF₀ diagrams syntax with single data elements. It also checks the consistencies of pipeline data elements to a limited extent. The overall function of this syntax checking expert system is achieved. The implementation of syntax checking expert system into SAtool II is proved feasible.

Recommendations

Based on the results and experiences of this study, this section presents some recommendations for future research which could lead to enhance the capability of both the Essential Model and the Syntax Expert System.

Boundary Single Data Item. The rules developed here so far to check single data items between parent and child activities has been successful. It is not likely at this point that there is a possibility to simplify the rules that already exist as a result of this research. If a different tool does, than it should be used in order to simplify the rule base so it will be easier to understand and be more efficient.

Boundary Pipeline Data Items. As pipelines could be a very complex combination, not all features are included here. More rules might be developed to feature the intermediate pipeline

data items, and check the consistency between parent activity and child activities where levels of pipeline data item relations might exist.

Further IDEF₀ Diagrams Drawing Features. Some features of the IDEF₀ diagrams which are not included in the Essential Model created in (16) and thus not considered in the syntax checking rules are also suggested:

- Expand the functions of the Essential Model to include tunnel arrows. Develop rules to check the consistency of those features of drawing information.

For instance, since a tunnel arrow does not have the information of the relationships between the attaching activity and its parent box. A tunnel arrow attaching an activity should have an ID to show that this arrow is a tunnel arrow. Thus the checking of consistency should check whether a missing boundary data element between a parent and its child diagrams is a tunnel, if it is, then it is not a boundary syntax error.

- Establish a mechanism either in Essential Model or in Drawing Model to represent squiggles, double arrows (feedback, and iteration). Expand syntax checking rules to include knowledge about the specific application being developed.
- Modify the menu selection interface such that the user could make the selection directly from the screen with a mouse.
- Use compiled rule base to improve efficiency of the syntax checking expert system.

Bibliography

1. Baker, Louis. *Artificial Intelligence With Ada*. New York: McGraw-Hill Publishing Company, Inc., 1989.
2. Bell Atlantic Knowledge Systems, Inc. *AI Get Real*. AI Expert. P.O. Box 3528 Princeton, New Jersey 08543-3528, 1991.
3. Booch, Grady. *Software Components with Ada*. Menlo Park, California: The Benjamin/Cummings Publishing Company, Inc., 1987.
4. Booch, Grady. *Software Engineering with Ada*. 2727 Sand Hill Road Menlo Park, CA 94025: The Benjamin/Cummings Publishing Company, Inc., 1987.
5. Bowles, Adrien J. "A Note on the Yourdon Structured Method," *ACM Software Engineering Notes*, 15(2):27 (April 1990).
6. Brassard, G. and P. Bratley. *Algorithmics, Theory and Practice*. Englewood Cliffs, New Jersey 07632: Prentice Hall, 1988.
7. Citrenbaum, Ronald and James R. Geissman. "Selecting a Shell," *AI Expert*, 1(1):21-26 (September 1987).
8. Cohen, Norman H. *Ada As a Second Language*. New York: McGraw-Hill Book Company, 1986.
9. Davis, Alan M. *Software Requirements Analysis and Specification*. Englewood Cliffs, New Jersey 07632: Prentice Hall, Inc., 1990.
10. Giarratano, Joseph. *Expert Systems*. 20 Park Plaza Boston, Massachusetts 02116: PWS-KENT Publishing Company, 1989.
11. Humphrey, Watts S. *Managing the Software Process*. Menlo Park, California: Addison-Wesley Publishing Company, 1990.
12. IntelliCorp. *IntelliCorp KEE Software Development System User's Manual* (3.3 Edition), 1986.
13. Johnson, Steven E. *A Graphics Editor for Structured Analysis with a Data Dictionary*. MS thesis, AFIT/GE/ENG/87D-128, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A190618).
14. Jung, Capt(ROKAF) Donghak H. *Design of a Syntax Validation Tool for Requirements Analysis Using Structured Analysis and Design Technique(SADT)*. MS thesis, AFIT/GCS/ENG/88S-1, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1988 (AD-A202725).
15. Kim, Capt(ROKAF) Intaek. *Expert System in Software Engineering Using Structured Analysis and Design Technique(SADT)*. MS thesis, AFIT/GCS/ENG/90J-2, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1990 (AD-A??????).
16. Kitchen, Terry LeVere Captain USAF. *An Object Oriented Design and Implementation For The IDEF₀ Essential Data Model with An Ada Based Expert System*. MS thesis, AFIT/GCS/ENG/90D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990 (AD-A230814).
17. Korth, Henry F. and Abraham Silberschatz. *Database System Concepts*. New York: McGraw-Hills, Inc, 1991.

18. Luger, George F. *Artificial Intelligence and the Design of Expert Systems*. 890 Bridge Parkway Redwood City, California 94065: The Benjamin/Cummings Publishing Company, Inc., 1989.
19. Marca, David A. and Clement L. McGowan. *SADT Structured Analysis and Design Technique*. New York: McGraw-Hill Book Company., 1988.
20. Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson AFB, OH 45433. *Integrated Computer-Aided Manufacturing (ICAM) Function Modeling Manual (IDEF₀)*, June 1981. Contract F33615-78-C-5158 with SofTech, Inc.
21. NASA - Johnson Space Center - Artificial Intelligence Section. *CLIPS Reference Manual: Version 4.3 of CLIPS*, July 1989.
22. NASA - Johnson Space Center - Artificial Intelligence Section. *CLIPS User's Guide: Version 4.3 of CLIPS*, August 1989.
23. Pearl, J. *Heuristics*. Menlo Park, California: Addison-Wesley Publishing Co., 1984.
24. Ross, Douglas T. "Structured Analysis (SA) : A Language for Communicating Ideas," *IEEE Transactions on Software Engineering*, SE-3(1):27 (April 1990).
25. Shlaer, Sally and Stephen J. Mellor. *Object-Oriented Systems Analysis*. Englewood Cliffs, New Jersey 07632: Prentice Hall, Inc., 1988.
26. Sommerville, Ian. *Software Engineering*. Menlo Park, California: Addison-Wesley Publishing Company, 1989.
27. Sommerville, Ian. *Software Engineering: Third Edition*. Reading MA: Addison-Wesley Publishing Company., 1989.
28. Tevis, Jay-Evan J. *Machine-Independent Ada Windows and Enhanced Graphics for SATool II.* MS thesis, AFIT/GCS/ENG/90D-?, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990 (AD-A2331239).
29. Tsai, Jeffrey J. P. and J. C. Ridge. "Intelligent Support for Specification Transformation," *IEEE Software*, 3(6):28-35 (December 1988).
30. Yourdon, Edward. *Modern Structured Analysis*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1989.

AFIT/GCS/ENG/91-J

AN ADA BASED EXPERT SYSTEM
FOR
THE ADA VERSION OF SATool II
VOLUME II: APPENDICES

THESIS

Min-fuh Shyong
Major, ROCAF

AFIT/GCS/ENG/91-J

AFIT/GCS/ENG/91-J

AN ADA BASED EXPERT SYSTEM
FOR
THE ADA VERSION OF SAtool II
VOLUME II: APPENDICES

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Computer Systems)

Min-fuh Shyong, B.S.C.S.
Major, ROCAF

DECEMBER, 1990

Approved for public release; distribution unlimited

Table of Contents

	Page
Table of Contents	ii
Appendix A. CLIPS BEHAVIOR IN THE BLOCKS WORLD PROBLEM	A-1
Appendix B. ESSENTIAL FACT UTILITIES	B-1
Appendix C. CLIPS RULE BASE	C-1
Appendix D. SAMPLE ESSENTIAL MODEL IDEF ₀ SYNTAX CHECKING SCRIPT FILE	D-1

Appendix A. *CLIPS BEHAVIOR IN THE BLOCKS WORLD PROBLEM*

In this example, the two stack of blocks is represented as facts format in CLIPS. A single block may be stacked upon another block. the goal of a complex blocks world program would be rearrange the stacks of block into a goal cnfiguration with the minimum number of moves. For this purpose, two restrictions are made:

1. Only one primary goal is allowed and this goal can only be to move one block on top on another block. If the goal is to move block x on top of block y, then move all blocks (if any) on top of block x to the floor and all blocks (if any) on top of block y to the floor and then move block x on top of block y.
2. Any goal must not have already been achieved. That is, the goal cannot be to move block x on top of block y if block x is already on top of block y.

Now we follow the step by step method of building a program mentioned in Chapter 2.

First: writing pseudorules using English-like text. We can figure out that there are only four kinds of possible conditions needed to achieve our goal.

1. When both the top of x and y are clear, move x on top of y directly.
2. When something on top of x, then move something on top of floor first before move x on top of y.
3. When something on top of y, then move something on top of floor first before move x on top of y.
4. Move something on top of floor before the next move can be achieved.

In this case, x is considered as upper block and y is considered as lower block. Then the pseudo code can be written as:

RULE Move-Directly

IF The goal is to move block ?upper on top of block ?lower and
 block ?upper is the top block in its stack and
 block ?lower is the top block in its stack,
THEN Move block ?upper on top of block ?lower

RULE Clear-Upper-Block

IF The goal is to move Block ?x and
 block ?x is not the top block in its stack and
 block ?above is on top of block ?x,
THEN The goal is to move block ?above to the floor

RULE Clear-Lower-Block

IF The goal is to move another block on top of block ?x and
 block ?x is not the top block in its stack and
 block ?above is on top of block ?x,
THEN The goal is to move block ?above to the floor

RULE Move-To-Floor

IF The goal is to move block ?upper on top of the floor and
 block ?upper is the top block in its stack,
THEN Move block ?upper on top of the floor

Second: Based on the information given above, the blocks can be represented as stacks of blocks and translated into initial knowledge for the program. The setting of the blocks is illustrated in Figure n and its facts format is as follows:

```
(def facts initial-state
  (stack a b c)
  (stack d e f)
  (move-goal c on-top-of e)
  (stack) )
```

Finally: The pseudorules were translated to CLIPS rules using the facts as a guide for translation.

```
(defrule move-directly
```

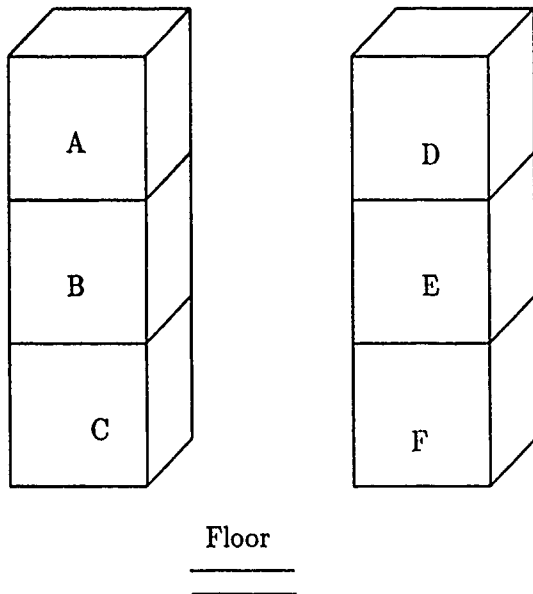


Figure A.1. Initial State of The Blocks World

```

?goal <- (move-goal ?b1 on-top-of ?b2)
?stack-1 <- (stack ?b1 $?rest1)
?stack-2 <- (stack ?b2 $?rest2)
=>
(retract ?goal ?stack-1 ?stack-2)
(assert (stack $?rest1))
(assert (stack ?b1 ?b2 $?rest2))
(fprintout t ?b1 " moved on top of " ?b2 "." crlf) )

(defrule move-to-floor
  ?goal <- (move-goal ?b1 on-top-of floor)
  ?stack-1 <- (stack ?b1 $?rest)
  =>
  (retract ?goal ?stack-1)
  (assert (stack ?b1))
  (assert (stack $?rest))
  (fprintout t ?b1 " moved on top of floor. " crlf) )

(defrule clear-upper-block
  (move-goal ?b1 on-top-of ?)
  (stack ?top $? ?b1 $?)
  =>
  (assert (move-goal ?top on-top-of floor)) )

(defrule clear-lower-block
  (move-goal ? on-top-of ?b1)
  (stack ?top $? ?b1 $?)

```

```
=>
(assert (move-goal ?top on-top-of floor)) )
```

Now lets see the results of running CLIPS for the Blocks World program. Any text after ';' will be coments added.

```
CLIPS> (load "a:Blocks-World.clp")
Processing deffacts block initial-state
Compiling rule: move-directly +j+j+j
Compiling rule: move-to-floor +j+j
Compiling rule: clear-upper-block =j+j
Compiling rule: clear-lower-block =j+j
CLIPS> (facts)
CLIPS> (reset)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (stack a b c)
f-2      (stack d e i)
f-3      (move-goal c on-top-of e)
f-4      (stack)
CLIPS> (rules)
move-directly
move-to-floor
clear-upper-block
clear-lower-block
CLIPS> (run)
a moved on top of floor.
b moved on top of flcor.
d moved on top of floor.
c moved on top of e.
7 rules fired
Run time is 1.5390625 seconds
CLIPS> (facts)
f-0      (initial-fact)
f-4      (stack)
f-6      (stack a)
f-9      (stack b)
f-12     (stack d)
f-14     (stack c e f)
CLIPS> (set-break move-directly)
CLIPS> (set-break move-to-floor)
CLIPS> (set-break clear-upper-block)
CLIPS> (set-break clear-lower-block)
CLIPS> (watch all)
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (stack a b c)
```



```

==> f-2      (stack d e f)
==> f-3      (move-goal c on-top-of e)
==> Activation 0      clear-lower-block: f-3,f-2
==> Activation 0      clear-upper-block: f-3,f-1
==> f-4      (stack)
CLIPS> (run)
FIRE      1 clear-upper-block: f-3,f-1
==> f-5      (move-goal a on-top-of floor)
==> Activation 0      move-to-floor: f-5,f-1
Breaking on rule move-to-floor
1 rules fired
Run time is 0.328125 seconds
CLIPS> (run)
FIRE      1 move-to-floor: f-5,f-1
<== f-5      (move-goal a on-top-of floor)
<== f-1      (stack a b c)
==> f-6      (stack a)
==> f-7      (stack b c)
==> Activation 0      clear-upper-block: f-3,f-7
a moved on top of floor.
Breaking on rule clear-upper-block
1 rules fired
Run time is 1.8203125 seconds
CLIPS> (run)
FIRE      1 clear-upper-block: f-3,f-7
==> f-8      (move-goal b on-top-of floor)
==> Activation 0      move-to-floor: f-8,f-7
Breaking on rule move-to-floor
1 rules fired
Run time is 0.3828125 seconds
CLIPS> (run)
FIRE      1 move-to-floor: f-8,f-7
<== f-8      (move-goal b on-top-of floor)
<== f-7      (stack b c)
==> f-9      (stack b)
==> f-10     (stack c)
b moved on top of floor.
Breaking on rule clear-lower-block
1 rules fired
Run time is 1.59375 seconds
CLIPS> (run)
FIRE      1 clear-lower-block: f-3,f-2
==> f-11     (move-goal d on-top-of floor)
==> Activation 0      move-to-floor: f-11,f-2
Breaking on rule move-to-floor
1 rules fired
Run time is 0.328125 seconds
CLIPS> (run)
FIRE      1 move-to-floor: f-11,f-2
<== f-11     (move-goal d on-top-of floor)
<== f-2      (stack d e f)

```

```

==> f-12    (stack d)
==> f-13    (stack e f)
==> Activation 0    move-directly: f-3,f-10,f-13
d moved on top of floor.
Breaking on rule move-directly
1 rules fired
Run time is 1.703125 seconds
CLIPS> (run)
FIRE    1 move-directly: f-3,f-10,f-13
<== f-3    (move-goal c on-top-of e)
<== f-10    (stack c)
<== f-13    (stack e f)
==> f-14    (stack c e f)
c moved on top of e.
1 rules fired
Run time is 0.4453125 seconds
CLIPS> (run)
0 rules fired
CLIPS> (facts)
f-0      (initial-fact)
f-4      (stack)
f-6      (stack a)
f-9      (stack b)
f-12     (stack d)
f-14     (stack c e f)
CLIPS> (agenda)
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (stack a b c)
==> f-2      (stack d e f)
==> f-3      (move-goal c on-top-of e)
==> Activation 0    clear-lower-block: f-3,f-2
==> Activation 0    clear-upper-block: f-3,f-1
==> f-4      (stack)

```

Appendix B. *ESSENTIAL FACT UTILITIES*

```
-----
-- DATE: 2/21/91 --
-- VERSION: 1.0 --
-- TITLE: Essential Subsystem Essential_Fact_Uilities Package --
-- FILENAME: es_factu.a --
-- COORDINATOR: Dr. Hartrum --
-- PROJECT: SATool II --
-- OPERATING SYSTEM: SUN OS Release 4.1 --
-- LANGUAGE: Verdex Ada Development System (VADS) - Version 5.41 --
-- FILE PROCESSING: Must be compiled after es_genev.a, es_proj.a, --
-- es_activ.a, es_datel.a, es_conof.a, es_ICOM.a, es_hista.a, --
-- es_calls.a --
-- CONTENTS: Package Essential_Fact_Uilities --
-- FUNCTIONS: This package contains two utility operations for each of--
-- the 7 packages that have a 'manager' in their names. --
-----
```

```
-----
-- SUMMARY OF RECENT MODIFICATIONS: --
-- 27 Oct 90: Added routines to retrieve and restore the project name.--
-- 10 Nov 90: Added routines to retrieve and restore a portion of the --
-- Activity Manager information. --
-- 5 Dec 90 : Added routines to retrieve the Historical Activity Facts--
-- 7 Dec 90 : Added routines to retrieve the Calls Relation Facts --
-- 8 Dec 90 : Added routines to retrieve the Consists of Relation Facts--
-- 8 Dec 90 : Added routines to retrieve the rest portion of the --
-- Activity Manager information. --
-- 11 Dec 90: Added routines to retrieve and restore the data --
-- element facts --
-- 21 Feb 91: All the Retrieve and Restore procedures tested and --
-- integrated with the Essential Model. --
-----
```

```
-----
-- DATE: 2/21/91 --
-- VERSION: 1.0 --
-- PACKAGE NAME: **ESSENTIAL FACT UTILITIES** --
-- LOCATED IN FILE: es_factu.a --
-- PURPOSE: This package is a collection of utility operations that --
-- interact with the managers. Each manager a 2 operations associ- --
-- ated with it: --
-- 1. An operation that retrieves either state information or --
-- information destined for CLIPS, based on a flag setting. --
-- 2. An operation that accepts state information as facts and --
-- restores that information to the manager data structure. --
-- Warning: The operations in this package depend heavily on the --
-- specific column numbers of the stored information. An alternative --
-----
```

```

-- to this methodology is to develop a parser to examine the fact --
-- strings. --
-- PACKAGE VISIBILITIES REQUIRED: Project_Manager, Activity_Manager, --
-- Data_Element_Manager, Consists_Of_Relation_Manager, ICOM_Relation_ --
-- Manager, Calls_Relation_Manager, Historical_Activity_Manager --
-- PACKAGE COMPOSITION: Specification and Body --
-- GENERICS INSTANTIATED: None --
-- ADT DESCRIPTION: N/A since this is a group of utilities. --
-- ORDER-OF: --
-- Visible: Retrieve_ICOM_Facts O(a * i) --
-- ( O(i) time when type_facts_flag = true ) --
-- Restore_ICOM_Facts O(i * i) --
-- Retrieve_Activity_Facts O(a * max(x, z)) --
-- Restore_Activity_Facts O(a * max(x, a * z) ) --
-- Retrieve_Project_Facts O(1) --
-- Restore_Project_Facts O(1) --
-- Hidden: Padded_String O(The_Size) --
-- where i is the number of icom relations, a is the number of --
-- activities, x is the number of lines in an activity description, --
-- and z is the number of children an activity has --
-- AUTHOR(S): Terry Kitchen and Min-fuh Shyong --
-- HISTORY: None (initial implementation) --

```

```

-----
with Text_IO;
with Activity_Class, Activity_Manager;
with Data_Element_Class, Project_Manager;
with ICOM_Relation_Class, ICOM_Relation_Manager;
with Environment_Types;

with Historical_Activity_Class, Historical_Activity_Manager;
with Calls_Relation_Class, Calls_Relation_Manager;
with Consists_Of_Relation_Class, Consists_Of_Relation_Manager;
with Data_Element_Class, Data_Element_Manager;

```

package Essential_Fact_Uilities is

```

-- Based on the type_facts_flag, the procedure passes a list of facts
-- to be stored in a file or a list of facts to be placed in an expert
-- system.
-- ***** 1 *****
procedure Retrieve_ICOM_Facts
(Type_Facts_Flag : in boolean;
Fact_Manager : in out Environment_Types.Fact_Buffer_Package.Manager_Type);

-- Takes an input buffer of ICOM facts and loads the information back
-- into the data structures.
procedure Restore_ICOM_Facts
(The_Fact_Buffer : in
Environment_Types.Fact_Buffer_Package.Manager_Type);

```

```

-- ***** 2 *****

procedure Retrieve_Project_Facts
(Type_Facts_Flag : in boolean;
 Fact_Manager : in out Environment_Types.Fact_Buffer_Package.Manager_Type);

-- Takes an input buffer of project fact(s) and loads the information back
-- into the data structure.

procedure Restore_Project_Facts
(The_Fact_Buffer : in
 Environment_Types.Fact_Buffer_Package.Manager_Type);

-- ***** 3 *****

procedure Retrieve_Activity_Facts
(Type_Facts_Flag : in boolean;
 Fact_Manager : in out Environment_Types.Fact_Buffer_Package.Manager_Type);

-- Takes an input buffer of activity facts and loads the information back
-- into the Activity_Manager. It must be modified to restore all the
-- activity facts once the Retrieve_Activity_Facts operation is completed.

procedure Restore_Activity_Facts
(The_Fact_Buffer : in
 Environment_Types.Fact_Buffer_Package.Manager_Type);

-- The project manager at this time only stores a single name; however,
-- in the future it could hold multiple projects.

-- ***** 4 *****

procedure Retrieve_Data_Element_Facts
(Type_Facts_Flag : in boolean;
 Fact_Manager : in out
 Environment_Types.Fact_Buffer_Package.Manager_Type) ;

procedure Restore_Data_Element_Facts
(The_Fact_Buffer : in
 Environment_Types.Fact_Buffer_Package.Manager_Type) ;

-- ***** 5 *****

procedure Retrieve_Historical_Activity_Facts
(Type_Facts_Flag : in boolean;
 Fact_Manager : in out
 Environment_Types.Fact_Buffer_Package.Manager_Type) ;

```

```

procedure Restore_Historical_Activity_Facts
  (The_Fact_Buffer: in
   Environment_Types.Fact_Buffer_Package.Manager_Type) ;

-- ***** 6 *****

procedure Retrieve_Calls_Relation_Facts
  (Type_Facts_Flag : in boolean;
   Fact_Manager    : in out
   Environment_Types.Fact_Buffer_Package.Manager_Type) ;

procedure Restore_Calls_Relation_Facts
  (The_Fact_Buffer : in
   Environment_Types.Fact_Buffer_Package.Manager_Type) ;

-- ***** 7 *****

procedure Retrieve_Consists_Of_Relation_Facts
  (Type_Facts_Flag : in boolean;
   Fact_Manager    : in out
   Environment_Types.Fact_Buffer_Package.Manager_Type) ;

procedure Restore_Consists_Of_Relation_Facts
  (The_Fact_Buffer : in
   Environment_Types.Fact_Buffer_Package.Manager_Type) ;

-----
end Essential_Fact_Uilities;
=====

package body Essential_Fact_Uilities is

-----
-- DATE: 10/23/90
-- VERSION: 1.0
-- NAME:      *****FUNCTION PADDED STRING*****
-- MODULE NUMBER: TBD
-- DESCRIPTION: A utility function to pad blanks to the front of a
-- of a string to reach a desired size.
-- ALGORITHM: A simple if then else with one embedded loop construct.
-- PASSED VARIABLES: The_String, The_Size
-- RETURNS: string
-- GLOBAL VARIABLES USED: None
-- GLOBAL VARIABLES CHANGED: None
-----

```

```

-- FILES READ: None
-- FILES WRITTEN: None
-- HARDWARE INPUT: None
-- HARDWARE OUTPUT: None
-- MODULES CALLED: None
-- CALLING MODULES: TBD
-- ORDER-OF: O(The_Size)
-- Note: that all slice operations are modeled as O(1) time.
-- AUTHOR(S): Terry Kitchen
-- HISTORY: None (Initial Implementation)
-----

```

```

function Padded_String(The_String : in string ; The_Size : in natural)
    return string is
-- Local Declarations --
Result_String: string(1..The_Size);
Start_Position: natural;
begin
if The_Size <= The_String'length then
    Start_Position:= The_String'length - The_Size + 1;
    -- Slice operation is modeled as O(1) time.
    Result_String:= The_String(Start_Position..The_String'length);
else
    Start_Position:= The_Size - The_String'length + 1;
    Result_String(Start_Position..The_Size):= The_String;
    -- worst case here - start_position is (The_Size - 2)
    -- Thus, O(The_Size) time in the worst case.
    for i in 1..(Start_Position - 1) loop
        Result_String(i):= ' ';
    end loop;
end if;
return Result_String;
end Padded_String;

```

```

-----
-- DATE: 10/23/90
-- VERSION: 1.0
-- NAME:          ***PROCEDURE RETRIEVE ICOM FACTS***
-- MODULE NUMBER: TBD
-- DESCRIPTION:
--      When the flag Type_Of_Facts_Flag is set to true, it means the
--      client procedure wants all the facts that are necessary for
--      the .esm file. If the flag is false, then the facts for
--      the expert system are returned. Facts of the same type have
--      the same format no matter where they are destined. In this
--      case, one extra type of fact is returned if the destination
--      is the an expert system (icom count fact).

```

```

-- icom tuple facts: (retrieved when creating a .esm file or when --
-- performing check syntax) --
-- 1) a predefined attribute name (icom-tuple) --
-- 2) an activity name --
-- 3) a data element name --
-- 4) an icom code (i,c,o, or m) --
-- 5) and id number (an integer) --
-- icom count facts: (retrieved only when destination is CLIPS) --
-- 1) a predefined attribute name (e.g., icom-activity-inputs) --
-- 2) an activity name --
-- 3) an integer number representing the input count e.g. --
-- ALGORITHM: One while loop extracts the ICOM facts. A second loop --
-- which contains an O(i) procedure call is used to extract additional--
-- facts based on the contents of two different managers. --
-- PASSED VARIABLES: Type_Facts_Flag, Fact_Manager --
-- RETURNS: None --
-- GLOBAL VARIABLES USED: None --
-- GLOBAL VARIABLES CHANGED: None --
-- FILES READ: None --
-- FILES WRITTEN: None --
-- HARDWARE INPUT: None --
-- HARDWARE OUTPUT: None --
-- MODULES CALLED: None --
-- CALLING MODULES: TBD --
-- ORDER-OF: O(a * i) where 'a' is the number of activities and 'i' --
-- is the number of tuples in the ICOM relation manager. --
-- Note that all slice operations are modeled as O(1) time. --
-- AUTHOR(S): Terry Kitchen --
- HISTORY: None (Initial Implementation) --

```

```

-----
procedure Retrieve_ICOM_Facts
  (Type_Facts_Flag : in boolean;
   Fact_Manager    : in out Environment_Types.
                        Fact_Buffer_Package.Manager_Type) is

-- Local Declarations --
Fact_Pointer: Environment_Types.Fact_Buffer_Package.Iterator_Type;
A_Fact: Environment_Types.Fact_String_Type;
Blank_Fact: Environment_Types.Fact_String_Type:= (others => ' ');

-- Activity Related Declarations --
Act_Name: Activity_Class.Activity_Name_Type;
The_Act_Record: Activity_Class.Activity_Record_Type;

-- ICOM Related Declarations --
ICOM_Relation_Record: ICOM_Relation_Class.ICOM_Relation_Record_Type;
ICOM_Relation_Pointer: ICOM_Relation_Manager.ICOM_Relation_Pointer_Type;
ICOM_Code: character; -- a character 'i', 'c', 'o', or 'm'.
ICOM_Pair_Id: natural;
Inputs, Outputs, Controls, Mechanisms : natural:= 0;

```



```

begin
-- Clear the passed in fact_manager.
Environment_Types.Fact_Buffer_Package.Clear(Fact_Manager);

-- Retrieve the state information from the tuples regardless of the
-- flag setting.
-- Set pointer to beginning of manager.
ICOM_Relation_Manager.Reset_ICOM_Relation_Tuple_Iterator;

-- Engage O(i) loop to extract the icom_tuple facts. The facts
-- extracted will have the format discussed above. If there are
-- no ICOM tuples, this loop won't execute and an empty buffer is the
-- result.
While not ICOM_Relation_Manager.ICOM_Relation_Tuple_Iterator_Done loop
-- Get a record.
ICOM_Relation_Record:= ICOM_Relation_Manager.
                        Value_Of_ICOM_Relation_Tuple_At_Iterator;
-- Place the record into a fact string at specific positions.
-- Initialize the fact string to all blanks first.
-- All string assignments are modeled as O(1).
A_Fact:= Blank_Fact;
A_Fact(1..10) := "icom-tuple";
A_Fact(11)    := ' ';
A_Fact(12..36) := ICOM_Relation_Record.Activity;
A_Fact(37)    := ' ';
A_Fact(38..62) := ICOM_Relation_Record.Data_Element;
A_Fact(63)    := ' ';
A_Fact(64)    := ICOM_Relation_Record.Relationship;
A_Fact(65)    := ' ';
A_Fact(66..71) :=
    Padded_String(integer'image(ICOM_Relation_Record.Pair_Id), 6);

-- Store this fact in the fact buffer.
Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);
-- Advance pointer to next ICOM tuple in manager.
ICOM_Relation_Manager.Advance_Iterator_To_Next_ICOM_Relation_Tuple;
end loop;

-- Facts for expert system only.
-- Perform check here to determine if ICOM counts are requested.
if Type_Facts_Flag = False then
-- For each activity, need to determine the number of inputs, outputs,
-- controls and mechanisms. So, engage loop to get an
-- activity name then use the name to determine the ICOM counts.
-- Loop executes a times with an O(i) procedure call. Thus, the
-- order-of is O(a * i).
Activity_Manager.Reset_Activity_Iterator;
while not Activity_Manager.Activity_Iterator_Done loop
-- Get an activity record that contains a name.
The_Act_Record:= Activity_Manager.Value_Of_Activity_At_Iterator;

```

```

-- O(i) procedure to count the "arrows" for this activity.
-- Note: if the ICOM mgr is empty, this procedure returns all
-- zeroes and does not examine the activity name.
ICOM_Relation_Manager.Value_Of_ICOM_Counts
    (The_Act_Record.Name, Inputs, Controls, Outputs, Mechanisms);

-- Now must add the facts. (a better block of code is possible here)
-- Place the record into a fact string at specific positions.
-- Initialize the fact string to all blanks first.
-- All string assignments are modeled as O(1).
-- Add fact for number of inputs.
A_Fact:= Blank_Fact;
-- The padding of blanks in the first field is for aesthetic
-- purposes only.
A_Fact(1..24) := "icom-activity-inputs  ";
A_Fact(25)    := ' ';
A_Fact(26..50) := The_Act_Record.Name;
A_Fact(51)    := ' ';
A_Fact(52..57) := Padded_String(integer'image(Inputs), 6);
-- Store this fact in the fact buffer.  O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

-- Add fact for number of controls.  O(1) time.
A_Fact:= Blank_Fact;
A_Fact(1..24) := "icom-activity-controls ";
A_Fact(25)    := ' ';
A_Fact(26..50) := The_Act_Record.Name;
A_Fact(51)    := ' ';
A_Fact(52..57) := Padded_String(integer'image(Controls), 6);
-- Store this fact in the fact buffer.
Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

-- Add fact for number of outputs.  O(1) time.
A_Fact:= Blank_Fact;
A_Fact(1..24) := "icom-activity-outputs  ";
A_Fact(25)    := ' ';
A_Fact(26..50) := The_Act_Record.Name;
A_Fact(51)    := ' ';
A_Fact(52..57) := Padded_String(integer'image(Outputs), 6);
-- Store this fact in the fact buffer.
Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

-- Add fact for number of mechanisms.  O(1) time.
A_Fact:= Blank_Fact;
A_Fact(1..24) := "icom-activity-mechanisms";
A_Fact(25)    := ' ';
A_Fact(26..50) := The_Act_Record.Name;
A_Fact(51)    := ' ';

```

```

    A_Fact(52..57) := Padded_String(integer'image(Mechanisms), 6);
    -- Store this fact in the fact buffer.
    Environment_Types.Fact_Buffer_Package.Add_Item
        (A_Fact, Fact_Manager, Fact_Pointer);

    -- Advance Activity_Manger iterator to next activity record.
    Activity_Manager.Advance_Iterator_To_Next_Activity;
end loop;
end if;

end Retrieve_ICOM_Facts;

-----
-- DATE: 10/02/90
-- VERSION: 1.0
-- NAME:      ***PROCEDURE RESTORE ICOM FACTS***
-- MODULE NUMBER: TBD
-- DESCRIPTION: Restores the icom facts into the ICOM Relation Manager
-- ALGORITHM: A single while loop controls the execution with an
-- embedded call to an O(i) procedure.
-- PASSED VARIABLES: The_Fact_Buffer (contains the icom facts)
-- RETURNS: None
-- GLOBAL VARIABLES USED: None
-- GLOBAL VARIABLES CHANGED: None
-- FILES READ: None
-- FILES WRITTEN: None
-- HARDWARE INPUT: None
-- HARDWARE OUTPUT: None
-- MODULES CALLED: None
-- CALLING MODULES: TBD
-- ORDER-OF: O(i * i) where i is the number of facts in the fact
-- buffer which should be the same as the no. of ICOM tuples; the
-- of ICOM tuples is represented by an 'i'.
-- Note that all string slice operations are modeled as O(1) time.
-- AUTHOR(S): Terry Kitchen
-- HISTORY: None (Initial Implementation)
-----

procedure Restore_ICOM_Facts
(The_Fact_Buffer : in
    Environment_Types.Fact_Buffer_Package.Manager_Type) is

    -- Local Declarations --

    Fact_Pointer: Environment_Types.Fact_Buffer_Package.Iterator_Type;
    A_Fact: Environment_Types.Fact_String_Type;
    First_Char: natural:= 0;
    Char_Position: natural:= 0;
    Temp_Pos: natural:= 0;

    -- ICOM Related Declarations --
    ICOM_Relation_Record: ICOM_Relation_Class.ICOM_Relation_Record_Type;

```

```

ICOM_Relation_Pointer: ICOM_Relation_Manager.ICOM_Relation_Pointer_Type;
Null_ICOM_Record: ICOM_Relation_Class.ICOM_Relation_Record_Type;

begin
-- Check for empty buffer of facts. If empty, do nothing.
if Environment_Types.Fact_Buffer_Package.Is_Empty(The_Fact_Buffer) then
    return;
end if;

-- Initialize iterator to first tuple/fact.
Environment_Types.Fact_Buffer_Package.Initialize_Iterator
    (Fact_Pointer, The_Fact_Buffer);

-- Engage loop to extract the icom_tuple facts from a buffer
-- one at a time. This loop is O(i) time.
While not Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) loop
    -- Get a record.
    A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);
    -- Since we put the information in the string, we know the
    -- exact columns where information should be.
    -- All string assignments are modeled as O(1);
    -- Insure the fields are all blanks.
    ICOM_Relation_Record:= Null_ICOM_Record;

    -- Retrieve the fields from the fact string.
    ICOM_Relation_Record.Activity:= A_Fact(12..36);
    ICOM_Relation_Record.Data_Element:= A_Fact(38..62);
    ICOM_Relation_Record.Relationship:= A_Fact(64);
    ICOM_Relation_Record.Pair_Id:= integer'value(A_Fact(66..71));

    -- Load this fact back into the ICOM manager. O(i) procedure call.
    ICOM_Relation_Manager.Create_ICOM_Relation_Tuple
        (ICOM_Relation_Record, ICOM_Relation_Pointer);

    -- Advance pointer to next ICOM tuple in manager.
    Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
end loop;
end Restore_ICOM_Facts;

```

```

-----
-- DATE: 10/27/90
-- VERSION: 1.0
-- NAME:          ***PROCEDURE RETRIEVE PROJECT FACTS***
-- MODULE NUMBER: TBD
-- DESCRIPTION:
--      When the flag Type_Of_Facts_Flag is set to true, it means the
--      client procedure wants all the facts that are necessary for
--      the .esm file. If the flag is false, then the facts for

```

```

--      the expert system are returned. Facts of the same type have --
--      the same format no matter where they are destined. In this --
--      case, the project name is but a single fact. Future --
--      modifications to SAtool II could include more information in --
--      the Project_Manager however, thus this procedure is of use. --
-- icom tuple facts: (retrieved when creating a .esm file or when --
--                    performing check syntax) --
-- 1) a predefined attribute name (project-name) --
-- 2) the project name (if the name is null, the word 'null' is --
--    placed in the field. --
-- ALGORITHM: All simple O(1) statements and 2 O(1) procedure calls. --
-- PASSED VARIABLES: Type_Facts_Flag, Fact_Manager --
-- RETURNS: None --
-- GLOBAL VARIABLES USED: None --
-- GLOBAL VARIABLES CHANGED: None --
-- FILES READ: None --
-- FILES WRITTEN: None --
-- HARDWARE INPUT: None --
-- HARDWARE OUTPUT: None --
-- MODULES CALLED: None --
-- CALLING MODULES: TBD --
-- ORDER-OF: O(1) --
-- AUTHOR(S): Terry Kitchen --
-- HISTORY: None (Initial Implementation) --

```

```

-----
procedure Retrieve_Project_Facts
  (Type_Facts_Flag : in boolean;
   Fact_Manager    : in out Environment_Types.
                        Fact_Buffer_Package.Manager_Type) is

```

```

-- Local Declarations --

```

```

Fact_Pointer: Environment_Types.Fact_Buffer_Package.Iterator_Type;
A_Fact: Environment_Types.Fact_String_Type;
Blank_Fact: Environment_Types.Fact_String_Type:= (others => ' ');

```

```

-- Project Related Declarations --

```

```

Project_Name: Environment_Types.Project_Name_Type;

```

```

begin

```

```

-- Clear the passed in fact_manager.

```

```

Environment_Types.Fact_Buffer_Package.Clear(Fact_Manager);

```

```

-- For the Project Manager, the same information is returned

```

```

-- regardless of the flag setting. The parameters are still used

```

```

-- however in case future modifications will need them.

```

```

Project_Name:= Project_Manager.Value_Of_Project_Name;

```

```

-- Check for blank name. If it's blank, give it the name 'null'.

```

```

-- The then part should never execute if SAtool II forces the user

```

```

-- to always assign a name to a project.

```

```

if Project_Name = Environment_Types.Null_Project_Name then

```

```

    Project_Name:= "null";
end if;

-- Create the fact. All O(1) time.
A_Fact:= Blank_Fact;
A_Fact(1..12):= "project-name";
A_Fact(13):= ' ';
A_Fact(14..38):= Project_Name;

-- Store this fact in the fact buffer. Just one fact. No loop.
-- O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

end Retrieve_Project_Facts;

-----
-- DATE: 10/27/90
-- VERSION: 1.0
-- NAME:      ***PROCEDURE RESTORE PROJECT FACTS***
-- MODULE NUMBER: TBD
-- DESCRIPTION:
-- ALGORITHM: All O(1) statements and procedure calls.
-- PASSED VARIABLES: The_Fact_Buffer (contains the project fact)
-- RETURNS: None
-- GLOBAL VARIABLES USED: None
-- GLOBAL VARIABLES CHANGED: None
-- FILES READ: None
-- FILES WRITTEN: None
-- HARDWARE INPUT: None
-- HARDWARE OUTPUT: None
-- MODULES CALLED: None
-- CALLING MODULES: TBD
-- ORDER-OF: O(1)
-- Note that all string slice operations are modeled as O(1) time.
-- AUTHOR(S): Terry Kitchen
-- HISTORY: None (Initial Implementation)
-----

procedure Restore_Project_Facts
    (The_Fact_Buffer : in
     Environment_Types.Fact_Buffer_Package.Manager_Type) is
-- Local Declarations --
Fact_Pointer: Environment_Types.Fact_Buffer_Package.Iterator_Type;
A_Fact: Environment_Types.Fact_String_Type;
First_Char: natural:= 0;

-- Project Related Declarations --
Project_Name: Environment_Types.Project_Name_Type;

begin
-- Check for empty buffer of facts. If empty, do nothing.

```

```

if Environment_Types.Fact_Buffer_Package.Is_Empty(The_Fact_Buffer) then
    return;
end if;

-- Initialize iterator to first tuple/fact.
Environment_Types.Fact_Buffer_Package.Initialize_Iterator
    (Fact_Pointer, The_Fact_Buffer);

-- Get a record. There is only one fact! O(1) call.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

-- We know that the project name starts in column 14 at this point.
First_Char:= 14;

-- Since there is only one field of data, no looping is necessary.
-- The remaining characters after the project name are blank.
Project_Name:= A_Fact(First_Char..38);

-- Need to check if the project name was null.
-- If the name is null, we do nothing, since the Project_Manager
-- initializes the project name to all blanks anyway.
if Project_Name(1..4) = "null" then
    null;
else
    -- Load the project name back into the manager.
    Project_Manager.Set_Project_Name(Project_Name);
end if;
end Restore_Project_Facts;

```

```

-----
-- DATE: 2/19/91 --
-- VERSION: 1.0 --
-- NAME: ***PROCEDURE RETRIEVE ACTIVITY FACTS*** --
-- MODULE NUMBER: TBD --
-- DESCRIPTION: --
--      When the flag Type_Of_Facts_Flag is set to true, it means the --
--      client procedure wants all the facts that are necessary for --
--      the .esm file. If the flag is false, then the facts for --
--      the expert system are returned. --
-- Note that this procedure only handles a subset of the activity --
-- facts: the activity name, number, and description. The remaining --
-- facts must still be retrieved! --
-- Activity facts format for .esm file: --
-- Note that if an activity name exists, then the other fields --
-- will be filled with a string called 'null' if they are empty. --
--      (act-name 'activity name') --
--      (act-numb 'activity name' 'activity number') --
--      (act-desc 'activity name' 'word1' 'word2' etc.) --

```

```

-- The last fact is repeated for each line of the description. --
--
-- Activity facts format for CLIPS: --
-- Note that CLIPS does NOT need to check the description. Therefore, --
-- just pass it a fact with 'null' or 'not-null' to save space in --
-- the working memory. --
--   (act-name 'activity name') --
--   (act-numb 'activity name' 'activity number') --
--   (act-desc 'activity name' 'not-null') --
-- As with the facts for the .esm file, if any of the fields are empty --
-- the word 'null' is inserted instead. --
-- ALGORITHM: One outer loop that iterates through the activities --
-- contains simple if then else constructs and one inner loop. These --
-- mechanisms contain several O(1) function calls to the activity --
-- manager to retrieve information. --
-- PASSED VARIABLES: Type_Facts_Flag, Fact_Manager --
-- RETURNS: None --
-- GLOBAL VARIABLES USED: None --
-- GLOBAL VARIABLES CHANGED: None --
-- FILES READ: None --
-- FILES WRITTEN: None --
-- HARDWARE INPUT: None --
-- HARDWARE OUTPUT: None --
-- MODULES CALLED: Several Activity Manager procedures and functions, --
-- plus some Text_Buffer_Package operations. --
-- CALLING MODULES: Essential_IO.Save_Project, Clips_Working_Memory_ --
-- Interface.Assert_All_Facts --
-- ORDER-OF:  $O(a * \max(x, z))$  where  $a$  = # activities, and  $x$  is --
-- the number of lines in a description, and  $z$  = # of activ children. --
-- Note that all slice operations are modeled as  $O(1)$  time. --
-- AUTHOR(S): Terry Kitchen and Min-fuh Shyong --
-- HISTORY: None (Initial Implementation) --

```

```

-----
procedure Retrieve_Activity_Facts
  (Type_Facts_Flag : in boolean;
   Fact_Manager    : in out Environment_Types.
                       Fact_Buffer_Package.Manager_Type) is

```

```

  -- Local Declarations --

```

```

Fact_Pointer: Environment_Types.Fact_Buffer_Package.Iterator_Type;
A_Fact: Environment_Types.Fact_String_Type;
Blank_Fact: Environment_Types.Fact_String_Type:= (others => ' ');

```

```

  -- Activity Related Declarations --

```

```

Act_Name      : Activity_Class.Activity_Name_Type;
The_Act_Record : Activity_Class.Activity_Record_Type;

The_Iterator   : Environment_Types.Text_Buffer_Package.Iterator_Type;
Child_Iterator : Environment_Types.Data_Buffer_Package.Iterator_Type;

```



```

A_Description_Line : Environment_Types.DD_Text_Type;
A_Child            : Environment_Types.DD_Field_Type;

-- **** Added new facts 120390 ****--
Reference_Iterator : Environment_Types.Text_Buffer_Package.Iterator_Type;
A_Reference_Line   : Environment_Types.DD_Text_Type;

-- **** Added new Vars for version changes ***

Version_Iterator   : Environment_Types.Text_Buffer_Package.Iterator_Type;
Version_Line       : Environment_Types.DD_Text_Type;

begin
-- Clear the passed in fact_manager. O(1) time.
Environment_Types.Fact_Buffer_Package.Clear(Fact_Manager);

-- Set pointer to beginning of manager. O(1) time.
Activity_Manager.Reset_Activity_Iterator;

-- Engage loop to extract the facts associated with an activity. The
-- facts extracted will have the format discussed above. If there are
-- no activities, this loop won't execute and an empty buffer is the
-- result.
-- This loop runs a times where a is the number of activities. At this
-- time there is only one inner loop of O(x) time. Thus, the time
-- complexity is O(a * x).

while not Activity_Manager.Activity_Iterator_Done loop
-- Get a record. O(1) time.
The_Act_Record:= Activity_Manager.
                    Value_Of_Activity_At_Iterator;

-- Regardless of the Type_Facts_Flag setting, the activity name is
-- always added to the fact buffer.

-- Place the activity name into a fact string at specific positions.
-- Initialize the fact string to all blanks first.
-- All string assignments are modeled as O(1).
-- ****Create Activity Name Fact****
A_Fact:= Blank_Fact;
A_Fact(1..8) := "act-name";
A_Fact(9)    := ' ';
A_Fact(10..34) := The_Act_Record.Name;

-- Store this fact in the fact buffer. O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

-- ****Create Activity Number Fact****

```

```

A_Fact:= Blank_Fact;
A_Fact(1..8) := "act-numb";
A_Fact(9)    := ' ';
A_Fact(10..34) := The_Act_Record.Name;
A_Fact(35)   := ' ';

-- This if then construct determines what goes into the last field.
-- If the activity number is not null then create a fact with the
-- activity number in it. Flag setting doesn't matter here.
if The_Act_Record.Number /= Activity_Class.Null_Activity_Number then
    -- All statements modeled as O(1) time.
    A_Fact(36..55) := The_Act_Record.Number;
else
    -- The activity number is null, so create a null fact for
    -- either the .esm file or the expert system. Again, the flag
    -- setting does not matter. All O(1) time.
    A_Fact(36..39) := "null";
end if;

-- Store this fact in the fact buffer. O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

-- ****Create one or more Activity Description Facts****
-- If the activity description is null then only a single fact is
-- created regardless of the flag setting.
if Environment_Types.Text_Buffer_Package.Is_Empty
    (The_Act_Record.Description) then

    -- Create a null fact.
    A_Fact:= Blank_Fact;
    A_Fact(1..8) := "act-desc";
    A_Fact(9)    := ' ';
    A_Fact(10..34) := The_Act_Record.Name;
    A_Fact(35)   := ' ';
    A_Fact(36..39):="null";

    -- Store this fact in the fact buffer. O(1) time.
    Environment_Types.Fact_Buffer_Package.Add_Item
        (A_Fact, Fact_Manager, Fact_Pointer);

-- A false flag setting means the fact is for the expert system.
-- For a description, we don't want the whole description in the
-- working memory, so just store a "not-null" string!
elsif Type_Facts_Flag = False then

    -- Create a fact that shows the description is not null.
    A_Fact:= Blank_Fact;
    A_Fact(1..8) := "act-desc";
    A_Fact(9)    := ' ';
    A_Fact(10..34) := The_Act_Record.Name;

```

```

A_Fact(35)      := ' ';
A_Fact(36..43):= "not-null";

-- Store this fact in the fact buffer.  O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
  (A_Fact, Fact_Manager, Fact_Pointer);

else
  -- At this point we know the flag is true which means the fact
  -- is to go to the .esm file.  However, there may be multiple lines
  -- in the description thus a loop is required.

  -- Set iterator to first line of description.
  Environment_Types.Text_Buffer_Package.Initialize_Iterator
    (The_Iterator, The_Act_Record.Description);

  -- Engage loop to get each line of the description and
  -- make it a fact.  This loop is O(x) time where x is the
  -- number of lines in the description.
  while not Environment_Types.Text_Buffer_Package.Is_Done
    (The_Iterator) loop

    -- Retrieve a single line of text.  O(1) time.
    A_Description_Line:= Environment_Types.Text_Buffer_Package.
      Value_Of_Item(The_Iterator);

    -- Create a fact representing a single line of the description.
    A_Fact:= Blank_Fact;
    A_Fact(1..8) := "act-desc";
    A_Fact(9)    := ' ';
    A_Fact(10..34) := The_Act_Record.Name;
    A_Fact(35)   := ' ';
    A_Fact(36..95):= A_Description_Line;

    -- Store this fact in the fact buffer.  O(1) time.
    Environment_Types.Fact_Buffer_Package.Add_Item
      (A_Fact, Fact_Manager, Fact_Pointer);

    -- Advance pointer by one to next line.
    Environment_Types.Text_Buffer_Package.Get_Next(The_Iterator);
  end loop;
end if;

-- If the activity child list is null then only a single fact is
-- created regardless of the flag setting.
if Environment_Types.Data_Buffer_Package.Is_Empty
  (The_Act_Record.Children) then

  -- Create a null fact.
  A_Fact:= Blank_Fact;

```

```

A_Fact(1..13) := "act-has-child";
A_Fact(14)    := ' ';
A_Fact(15..39) := The_Act_Record.Name;
A_Fact(40)    := ' ';
A_Fact(41..44) := "null";

-- Store this fact in the fact buffer.  O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
  (A_Fact, Fact_Manager, Fact_Pointer);

-- At this point, regardless of the flag setting, all the children
-- are put into the fact buffer.  Thus, the same facts go to
-- either the .esm file or the expert system.
else
  -- Set iterator to first child in list.
  Environment_Types.Data_Buffer_Package.Initialize_Iterator
    (Child_Iterator, The_Act_Record.Children);

  -- Engage loop to get each child and
  -- make it a fact.  This loop is O(z) time where z is the
  -- number of children.
  while not Environment_Types.Data_Buffer_Package.Is_Done
    (Child_Iterator) loop

    -- Retrieve a single line of text.  O(1) time.
    A_Child:= Environment_Types.Data_Buffer_Package.
      Value_Of_Item(Child_Iterator);

    -- Create a fact representing a single line of the description.
    A_Fact:= Blank_Fact;
    A_Fact(1..13) := "act-has-child";
    A_Fact(14)    := ' ';
    A_Fact(15..39) := The_Act_Record.Name;
    A_Fact(40)    := ' ';
    A_Fact(41..65) := A_Child;

    -- Store this fact in the fact buffer.  O(1) time.
    Environment_Types.Fact_Buffer_Package.Add_Item
      (A_Fact, Fact_Manager, Fact_Pointer);

    -- Advance pointer by one to next line.
    Environment_Types.Data_Buffer_Package.Get_Next(Child_Iterator);
  end loop;
end if;

-- Advance pointer to next activity in manager.
-- Activity_Manager.Advance_Iterator_To_Next_Activity;
-- Should This be at the end of Retrieve Activity??

-- *****
-- ***** Added new facts from Activity Reference Type 12/08/90 *****

```

```

-- ***** Author: Min-fuh Shyong *****
-- ***** Create Activity reference type facts *****
-- *****

    A_Fact      := Blank_Fact;
    A_Fact(1..12) := "act-ref-type";
    A_Fact(13)   := ' ';
    A_Fact(14..38) := The_Act_Record.Name;
    A_Fact(39)   := ' ';

    if The_Act_Record.Reference_Type /= Environment_Types.Null_Reference_Type then
        A_Fact(40..64) := The_Act_Record.Reference_Type;

    else
        A_Fact(40..43) := "null";
    end if;

Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

    -- (act-ref-type Name Reference_Type) or
    -- (act-ref-type Name null)

-- *** Create one or more activity reference facts -----

    -- if the activity reference is null then only a single fact is created
    -- regardless of the flag setting

    if Environment_Types.Text_Buffer_package.Is_Empty
        (The_Act_Record.Reference) then
        -- true it is empty

            A_Fact      := Blank_Fact;
            A_Fact(1..7) := "act-ref";
            A_Fact(8)   := ' ';
            A_Fact(9..33) := The_Act_Record.Name;
            A_Fact(34)   := ' ';
            A_Fact(35..38) := "null";

            -- (act-ref Name null)

Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

    elsif Type_Facts_Flag = false then

        A_Fact := Blank_Fact;
        A_Fact(1..7) := "act-ref";
        A_Fact(8) := ' ';
        A_Fact(9..33) := The_Act_Record.Name;

```

```

A_Fact(34) := ' ';
A_Fact(35..42) := "not-null";
:
:
-- (act-ref Name not-null)
Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);
:
else
-- flag is true the file to .esm file, may be multiple line of
-- reference so need a loop to get it
:
Environment_Types.Text_Buffer_Package.Initialize_Iterator
(Reference_Iterator, The_Act_Record.Reference);

-- Engage a loop to get each line of the reference and make it a fact

while not Environment_Types.Text_Buffer_Package.Is_Done
(Reference_Iterator) loop

A_Reference_Line :=
Environment_Types.Text_Buffer_Package.
Value_Of_Item(Reference_Iterator);

A_Fact := Blank_Fact;
A_Fact(1..7) := "act-ref";
A_Fact(8) := ' ';
A_Fact(9..33) := The_Act_Record.Name;
A_Fact(34) := ' ';
A_Fact(35..94) := A_Reference_Line;

Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

Environment_Types.Text_Buffer_Package.Get_Next(Reference_Iterator);

end loop;
end if;
-- (act-ref Name Reference-Line1)
-- (act-ref Name Reference-Line2)

-- ***** Create Activity Version facts *****

A_Fact := Blank_Fact;
A_Fact(1..11) := "act-version";
A_Fact(12) := ' ';
A_Fact(13..37) := The_Act_Record.Name;
A_Fact(38) := ' ';

if The_Act_Record.Version /= Activity_Class.Null_Activity_Version_number then
A_Fact(39..48) := The_Act_Record.Version;

```

```

else
    A_Fact(39..42) := "null";
end if;

Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

    -- (act-version Name Activity-Version) or
    -- (act-version Name null)

-- *** Create one or more activity Version-Changes facts -----

    -- if the activity version changes is null then only a single
    -- fact is created regardless of the flag setting

if Environment_Types.Text_Buffer_package.Is_Empty
    (The_Act_Record.Version_Changes) then
    -- true it is empty

        A_Fact      := Blank_Fact;
        A_Fact(1..11) := "act-ver-chg";
        A_Fact(12)   := ' ';
        A_Fact(13..37) := The_Act_Record.Name;
        A_Fact(38)   := ' ';
        A_Fact(39..42) := "null";

Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

elseif Type_Facts_Flag = false then

    A_Fact      := Blank_Fact;
    A_Fact(1..11) := "act-ver-chg";
    A_Fact(12) := ' ';
    A_Fact(13..37) := The_Act_Record.Name;
    A_Fact(38) := ' ';
    A_Fact(39..46) := "not-null";

Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

else
    -- version not empty show the version changes to .esm --
Environment_Types.Text_Buffer_Package.Initialize_Iterator
(Version_Iterator, The_Act_Record.Version_Changes);

    -- Engage a loop to get each version of changes and make it a fact

while not Environment_Types.Text_Buffer_Package.Is_Done

```

```

    (Version_Iterator) loop

Version_Line :=
    Environment_Types.Text_Buffer_Package.Value_Of_Item(Version_Iterator);

    A_Fact := Blank_Fact;
    A_Fact(1..11) := "act-ver-chg";
    A_Fact(12) := ' ';
    A_Fact(13..37) := The_Act_Record.Name;
    A_Fact(38) := ' ';
    A_Fact(39..98) := Version_Line;

-- (act-ver-chg Name null)
-- (act-ver-chg Name Version-changes)

Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

Environment_Types.Text_Buffer_Package.Get_Next(Version_Iterator);

end loop;
end if;

-- ****Create Activity Date Fact ****-----
    A_Fact:= Blank_Fact;
    A_Fact(1..8) := "act-date";
    A_Fact(9) := ' ';
    A_Fact(10..34) := The_Act_Record.Name;
    A_Fact(35) := ' ';

    if The_Act_Record.Date /= Environment_Types.Null_Date then
        A_Fact(36..43) := The_Act_Record.Date;

    else

        A_Fact(36..39) := "null";
    end if;

-- Store this date fact in the fact buffer. O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

-- (act-date Name mm/dd/yy)
-- (act-date Name null)

-- ****Create Activity Author Fact****-----
    A_Fact:= Blank_Fact;
    A_Fact(1..10) := "act-author";

```



```

    A_Fact(11)      := ' ';
    A_Fact(12..36) := The_Act_Record.Name;
    A_Fact(37)      := ' ';

-- This if then construct determines what goes into the last field.
-- If the activity author is not null then create a fact with the
-- activity author in it. Flag setting doesn't matter here.

if The_Act_Record.Author /= Environment_Types.Null_Author_Name then
    -- All statements modeled as O(1) time.
    A_Fact(38..62) := The_Act_Record.Author;
else
    -- The activity author is null, so create a null fact for
    -- either the .esm file or the expert system. Again, the flag
    -- setting does not matter. All O(1) time.
    A_Fact(38..41) := "null";
end if;

-- Store this fact in the fact buffer. O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

    -- Advance pointer to next activity in manager.
    Activity_Manager.Advance_Iterator_To_Next_Activity;

end loop;

-- Outer loop for Retrieve_Activity_Facts; *****

end Retrieve_Activity_Facts;

-----
-- DATE: 2/19/91                                     --
-- VERSION: 1.0                                         --
-- NAME:          ***PROCEDURE RESTORE ACTIVITY FACTS*** --
-- MODULE NUMBER: TBD                                   --
-- DESCRIPTION: This procedure accepts a buffer of activity facts and --
-- restores that information into the activity manager. Of special --
-- note is that the procedure assumes the facts are in the same order --
-- in which they were stored.                             --
-- ALGORITHM: A single while loop controls the execution with an --
-- embedded call to an O(i) procedure.                     --
-- PASSED VARIABLES: The_Fact_Buffer (contains the facts) --
-- RETURNS: None                                           --
-- GLOBAL VARIABLES USED: None                             --
-- GLOBAL VARIABLES CHANGED: None                         --
-- FILES READ: None                                         --
-- FILES WRITTEN: None                                     --
-- HARDWARE INPUT: None                                    --
-- HARDWARE OUTPUT: None                                   --

```

```

-- MODULES CALLED: None
-- CALLING MODULES: Essential_IO.Restore_Project
-- ORDER-OF: order of is  $O(a * \max(x, z * (a * z)))$  where a is the
-- number of activities, x is the number of lines in a description
-- and z is the number of children that an activity has. Note that
-- this order of may change when more of the activity manager facts
-- are restored.
-- Note that all string slice operations are modeled as  $O(1)$  time.
-- AUTHOR(S): Terry Kitchen and Min-fuh Shyong
-- HISTORY: None (Initial Implementation)
-----

procedure Restore_Activity_Facts
  (The_Fact_Buffer : in
   Environment_Types.Fact_Buffer_Package.Manager_Type) is
-- Local Declarations --
Fact_Pointer: Environment_Types.Fact_Buffer_Package.Iterator_Type;
A_Fact: Environment_Types.Fact_String_Type;
First_Char: natural:= 0;
Char_Position: natural:= 0;
Temp_Pos: natural:= 0;
More_Descriptions_Flag: boolean;

-- Activity Related Declarations --

Activity_Record: Activity_Class.Activity_Record_Type;
Activity_Pointer: Activity_Manager.Activity_Pointer_Type;
Null_Activity_Record: Activity_Class.Activity_Record_Type;

The_Iterator: Environment_Types.Text_Buffer_Package.Iterator_Type;

A_Description_Line: Environment_Types.DD_Text_Type;
A_Child: Environment_Types.DD_Field_Type;
A_Reference_Line : Environment_Types.DD_Text_Type;
Version_Line      : Environment_Types.DD_Text_Type;

Found_Flag: boolean:= False;
Result_Flag: boolean;

-- Exception --
-- This exception is declared here because the Essential IO package does
-- not check to see the facts are in any specific order.

Invalid_Fact_Sequence_For_Activity: exception;
Activity_Hierarchy_Error_During_Restore: exception;

-----

begin
-- Check for empty buffer of facts. If empty, do nothing.
if Environment_Types.Fact_Buffer_Package.Is_Empty(The_Fact_Buffer) then
  return;

```

```

end if;

-- Initialize iterator to first fact.
Environment_Types.Fact_Buffer_Package.Initialize_Iterator
(Fact_Pointer, The_Fact_Buffer);

-- Engage loop to extract the activity facts from a buffer
-- one at a time. This loop will execute a times -- once for
-- each activity. Note that there are many facts associated with
-- a single activity. This loop runs a times. The loop has one
-- inner loop of order x and one procedure call of (a * z). Thus,
-- order of is  $O(a * \max(x, z * (a * z)))$ 

While not Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) loop
  -- Get a record.
  A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);
  -- Since we put the information in the string, we know the
  -- exact columns where information should be.
  -- All string assignments are modeled as  $O(1)$ ;
  -- Insure the fields are all blanks.
  Activity_Record:= Null_Activity_Record;

  -- The first fact should be the name.
  if A_Fact(1..8) /= "act-name" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: act-name  ");
    raise Invalid_Fact_Sequence_For_Activity;
  end if;

  -- The activity name must be in columns 10 through 34.
  Activity_Record.Name:= A_Fact(10..34);

  -- Check to see if activity already exists.  $O(a)$  call.
  Activity_Manager.Activity_Exists(Activity_Record.Name,
    Activity_Pointer, Found_Flag);
  if Found_Flag = False then
    -- Do  $O(a * z)$  procedure call to create an activity.
    Activity_Manager.Create_Activity
      (Activity_Record.Name, Activity_Pointer);
  end if;

  -- Advance pointer to next fact in manager.  $O(1)$ .
  Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
  -- Get a fact.  $O(1)$  time.
  A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

```

```

-- This fact should be the activity number.
if A_Fact(1..8) /= "act-numb" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: act-numb  ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Columns 10 through 34 must be the same activity name.
if A_Fact(10..34) /= Activity_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: act-numb.Name  ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Columns 36 through 55 hold the activity number if it is
-- not null.
if A_Fact(36..39) = "null" then
    -- Do nothing, there was no activity number.
    null;
else
    -- Get the number.
    Activity_Record.Number:= A_Fact(36..55);
    -- Do O(1) procedure call to update the activity in the activity
    -- manager.
    Activity_Manager.Set_Activity_Number
        (Activity_Pointer, Activity_Record.Number);
end if;

-- Advance pointer to next fact in manager.
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
-- If the fact buffer is empty at this point there is an error
-- in the format.
if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: act-name Is_Done  ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Get a fact.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

-- The series of fact(s) should be the activity description.
-- There is at least one act-desc fact and possible more.
if A_Fact(1..8) /= "act-desc" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: act-desc  ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Columns 10 through 34 must be the same activity name.

```

```

if A_Fact(10..34) /= Activity_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: act-desc.Name    ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- If the description is null then we are done with this attribute.
-- Need only to advance the pointer by one for the outer loop.
if A_Fact(36..39) = "null" then
    -- There is no description for the activity, so just advance
    -- the fact pointer.
    -- Advance pointer to next fact in manager. O(1) time.
    Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

else
    -- There must be one or more lines in the description.
    -- This loop will run x times where x is the number of lines in the
    -- description.
    while A_Fact(1..8) = "act-desc" loop
        -- I realize this check is repetitive on the first iteration.
        -- Columns 10 through 34 must be the same activity name.
        -- O(1) time complexity.
        if A_Fact(10..34) /= Activity_Record.Name then
            Text_IO.Put_Line(A_Fact);
            Text_IO.Put_Line("I am Exp: act-desc.Name else    ");
            raise Invalid_Fact_Sequence_For_Activity;
        end if;

        -- Pull the description from the fact.
        A_Description_Line:= A_Fact(36..95);

        -- Add the description to the description part of the
        -- activity record. O(1) time.
        Environment_Types.Text_Buffer_Package.Add_Item
            (A_Description_Line,Activity_Record.Description,The_Iterator);

        -- Advance pointer to next fact in manager.
        Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

        -- If it is not empty get the next fact. O(1) time.
        if not Environment_Types.Fact_Buffer_Package.Is_Done
            (Fact_Pointer) then
            A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
                (Fact_Pointer);
        else
            - If this is the last fact of the last activity exit the
            -- loop.
            exit;
        end if;
    end loop;
end loop;

```

```

-- There were one or more lines in the description so now must
-- place them with the activity in the activity manager. O(1).
Activity_Manager.Set_Activity_Description
    (Activity_Pointer, Activity_Record.Description);

end if;

-- If the fact buffer is empty at this point there is an error
-- in the format. O(1) time. I know this because Retrieve_Activity_
-- Facts will at least put a null entry for no children.

if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exception: act-has-child Is_Done ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Get a fact.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

-- The series of fact(s) should be the activity description.
-- There is at least one act-desc fact and possible more.
if A_Fact(1..13) /= "act-has-child" then
    Text_IO.Put_Line("I am Exp:act-has-child ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Columns 15 through 39 must be the same activity name.
if A_Fact(15..39) /= Activity_Record.Name then
    Text_IO.Put_Line("I am Exp: act-has-child.Name ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- If the child list is null then we are done with this attribute.
-- Need only to advance the pointer by one for the outer loop.

if A_Fact(41..44) = "null" then
    -- There is no child list for the activity, so just advance
    -- the fact pointer.
    -- Advance pointer to next fact in manager. O(1) time.
    Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
else
    -- There must be one or more children.
    -- This loop will run z times where z is the number of children.
    while A_Fact(1..13) = "act-has-child" loop
        -- I realize this check is repetitive on the first iteration.

```

```

-- Columns 15 through 39 must be the same activity name.
-- O(1) time complexity.
if A_Fact(15..39) /= Activity_Record.Name then
    Text_IO.Put_Line("I am Exp: act-has-child whild    ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Pull a child from the fact.
A_Child:= A_Fact(41..65);

-- In order to add a child to a parent, the Activity Manager
-- requires that the child already exist as an activity.
-- Thus, must create the activity first if needed.

-- Check to see if activity already exists. O(a) call.
Activity_Manager.Activity_Exists(A_Child,
    Activity_Pointer, Found_Flag);
if Found_Flag = False then
    -- Do O(a * z) procedure call to create an activity.
    Activity_Manager.Create_Activity
        (A_Child, Activity_Pointer);
end if;

-- Do another O(a * z) procedure call to add this activity
-- to the parent's child list.
Activity_Manager.Add_Activity_Child(Activity_Record.Name,
    A_Child, Result_Flag);
-- Check results.
if Result_Flag = False then
    Text_IO.Put_Line("I am Exp: act-has-child flag    ");
    raise Activity_Hierarchy_Error_During_Restore;
end if;

-- Must now call Activity Exists again in order to reset the
-- pointer for any future operations. O(a) time.
Activity_Manager.Activity_Exists(Activity_Record.Name,
    Activity_Pointer, Found_Flag);

-- Advance pointer to next fact in manager.
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

-- If it is not empty get the next fact. O(1) time.
if not Environment_Types.Fact_Buffer_Package.Is_Done
    (Fact_Pointer) then
    A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_item
        (Fact_Pointer);
else
    -- If this is the last fact of the last activity exit the
    -- loop.
exit;
end if;

```

```

    end loop;
end if;

```

```

if Environment_Types.Fact_Buffer_Package.Is_done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: act-ref-type - Is_done ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

```

```

-- Advance pointer to next fact in manager. O(1).
-- Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
-- this will cause the fact get next act-ref fact ealier than
-- expected!!!
-- Get a fact. O(1) time.

```

```

A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
(Fact_Pointer);

```

```

if A_Fact(1..12) /= "act-ref-type" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp:act-ref-type ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

```

```

-- Columns 14 through 38 must be the same activity name.
if A_Fact(14..38) /= Activity_Record.Name then
    Text_IO.Put_Line("I am Exp: act-ref-type.name ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

```

```

-- Columns 40 through 64 hold the activity Reference Type if it is
-- not null.

```

```

if A_Fact(40..43) = "null" then
    -- Do nothing, there was no activity Reference Type.
    null;
else

```

```

    -- Get the Reference type

```

```

    Activity_Record.Reference_Type:= A_Fact(40..64);
    -- Do O(1) procedure call to update the activity in the activity
    -- manager.

```

```

    Activity_Manager.Set_Activity_Reference_Type
        (Activity_Pointer, Activity_Record.Reference_Type);
end if;

```



```

-- *****
-- ***** Restore Activity Reference Facts *****
-- *****
-- 13 Feb 91

    -- Advance pointer to next fact in manager.
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
    -- If the fact buffer is empty at this point there is an error
    -- in the format.
if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: act-ref Is_Done    ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

    -- Get a fact.

A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);

    -- The series of fact(s) should be the activity Reference.
    -- There is at least one act-ref fact and possible more.
if A_Fact(1..7) /= "act-ref" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: act-ref ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

    -- Columns 9 through 33 must be the same activity name.
if A_Fact(9..33) /= Activity_Record.Name then
    Text_IO.Put_Line("I am Exp: act-ref.Name 1 ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

    -- If the Reference is null then we are done with this attribute.
    -- Need only to advance the pointer by one for the outer loop.

if A_Fact(35..38) = "null" then
    -- There is no reference for the activity, so just advance
    -- the fact pointer.
    -- Advance pointer to next fact in manager. O(1) time.
    Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
else
    -- There must be one or more lines in the reference.
    -- This loop will run x times where x is the number of lines in the

```

```

-- reference.
while A_Fact(1..7) = "act-ref" loop
  -- I realize this check is repetitive on the first iteration.
  -- Columns 9 through 33 must be the same activity name.
  -- O(1) time complexity.
  if A_Fact(9..33) /= Activity_Record.Name then
    Text_IO.Put_Line("I am Exp: act-ref.Name 2  ");
    raise Invalid_Fact_Sequence_For_Activity;
  end if;

  -- Pull the reference from the fact.
  A_Reference_Line:= A_Fact(35..94);

  -- Add the reference to the reference part of the
  -- activity record. O(1) time.
  Environment_Types.Text_Buffer_Package.Add_Item
    (A_Reference_Line,Activity_Record.Reference,The_Iterator);

  -- Advance pointer to next fact in manager.
  Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

  -- If it is not empty get the next fact. O(1) time.
  if not Environment_Types.Fact_Buffer_Package.Is_Done
    (Fact_Pointer) then

    A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
      (Fact_Pointer);
  else
    -- If this is the last fact of the last activity exit the
    -- loop.
    -- Text_IO.Put_Line("I am Exp: act-ref else  ");
    -- raise Invalid_Fact_Sequence_For_Activity;
  end if;
end loop;

-- There were one or more lines in the description so now must
-- place them with the activity in the activity manager. O(1).

Activity_Manager.Set_Activity_Reference
  (Activity_Pointer, Activity_Record.Reference);

end if;

if Environment_Types.Fact_Buffer_Package.Is_done(Fact_Pointer) then
  Text_IO.Put_Line("I am Exp: act-version Is_done  ");
  raise Invalid_Fact_Sequence_For_Activity;
end if;

```

```

-- Advance pointer to next fact in manager. O(1).
    --Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
-- Get a fact. O(1) time.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

-- This fact should be the activity Version.

if A_Fact(1..11) /= "act-version" then
    Text_IO.Put_Line("I am Exp: act-version  ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Columns 13 through 37 must be the same activity name.
if A_Fact(13..37) /= Activity_Record.Name then
    Text_IO.Put_Line("I am Exp: act-version.Name  ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Columns 39 through 48 hold the activity version if it is
-- not null.
if A_Fact(39..42) = "null" then
    -- Do nothing, there was no activity version.
    null;
else
    -- Get the version.
    Activity_Record.Version:= A_Fact(39..48);
    -- Do O(1) procedure call to update the activity in the activity
    -- manager.
    Activity_Manager.Set_Activity_Version
        (Activity_Pointer, Activity_Record.Version);
end if;

-- *** get Activity Version Changes Facts *** -----

-- Advance pointer to next fact in manager.
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
-- If the fact buffer is empty at this point there is an error
-- in the format.

if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line("I am Exp: act-ver-chg Is_Done  ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

```

```

-- Get a fact.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);

-- The series of fact(s) should be the activity Version Changes.
-- There is at least one act-ver-chg fact and possible more.
if A_Fact(1..11) /= "act-ver-chg" then
    Text_IO.Put_Line("I am Exp: act-ver-chg  ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Columns 13 through 37 must be the same activity version name.
if A_Fact(13..37) /= Activity_Record.Name then
    Text_IO.Put_Line("I am Exp: act-ver-chg.Name  ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- If the version change is null then we are done with this attribute.
-- Need only to advance the pointer by one for the outer loop.

if A_Fact(39..42) = "null" then
    -- There is no version change for the activity, so just advance
    -- the fact pointer.
    -- Advance pointer to next fact in manager.  O(1) time.

    Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

else
    -- There must be one or more lines in the version changes.
    -- This loop will run x times where x is the number of times in the
    -- version change.

    while A_Fact(1..11) = "act-ver-chg" loop
        -- I realize this check is repetitive on the first iteration.
        -- Columns 13 through 37 must be the same activity name.
        -- O(1) time complexity.
        if A_Fact(13..37) /= Activity_Record.Name then
            Text_IO.Put_Line("I am Exp: act-ver-chg.Name in loop  ");
            raise Invalid_Fact_Sequence_For_Activity;
        end if;

        -- Pull the description from the fact.
        Version_Line:= A_Fact(39..98);

        -- Add the version change to the Version Changes part of the
        -- activity record. O(1) time.
    end loop;
end if;

```

```

Environment_Types.Text_Buffer_Package.Add_Item
  (Version_Line, Activity_Record.Version_Changes, The_Iterator);

-- Advance pointer to next fact in manager.

Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

-- If it is not empty get the next fact. O(1) time.
if not Environment_Types.Fact_Buffer_Package.Is_Done
  (Fact_Pointer) then
  A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);
else
  -- If this is the last fact of the last activity exit the
  -- loop.
  --Text_IO.Put_Line("I am Exp: act-ver-chg else  ");
  -- raise Invalid_Fact_Sequence_For_Activity;
exit;
end if;
end loop;

-- There were one or more lines in the version changes so now must
-- place them with the activity in the activity manager. O(1).
Activity_Manager.Set_Activity_Version_Comments
  (Activity_Pointer, Activity_Record.Version_Changes);

end if;

--*** get Activity Date Facts *** -----

if Environment_Types.Fact_Buffer_Package.Is_done(Fact_Pointer) then
Text_IO.Put_Line("I am Exp: act-data Is_done ");
  raise Invalid_Fact_Sequence_For_Activity; -- raised
end if;

-- 2/18/.2340 Get_Next(Fact_Pointer)
-- This will get author ealier
-- Advance pointer to next fact in manager. O(1).
-- Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
-- Get a fact. O(1) time.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
  (Fact_Pointer);

-- This fact should be the activity date.

if A_Fact(1..8) /= "act-date" then

```

```

        Text_IO.Put_Line(A_Fact);
        Text_IO.Put_Line("I am Exp: act-date ");
        raise Invalid_Fact_Sequence_For_Activity;
    end if;

-- Columns 10 through 34 must be the same activity name.
if A_Fact(10..34) /= Activity_Record.Name then
    Text_IO.Put_Line("I am Exp: act-data.Name ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Columns 36 through 43 hold the activity version if it is
-- not null.
if A_Fact(36..39) = "null" then
    -- Do nothing, there was no activity version.
    null;
else
    -- Get the date.

    Activity_Record.Date:= A_Fact(36..43);
    -- Do O(1) procedure call to update the activity in the activity
    -- manager.

    Activity_Manager.Set_Activity_Date
        (Activity_Pointer, Activity_Record.Date);
end if;

-- *** Get Activity Author Facts *** -----

if Environment_Types.Fact_Buffer_Package.Is_done(Fact_Pointer) then
    Text_IO.Put_Line("I am Exp: act-author for Is_done ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

    -- Advance pointer to next fact in manager. O(1).
    Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
    -- Get a fact. O(1) time.

A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

-- This fact should be the activity author.

if A_Fact(1..10) /= "act-author" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: act-author ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

```

```

-- Columns 12 through 36 must be the same activity name.
if A_Fact(12..36) /= Activity_Record.Name then
    Text_IO.Put_Line("I am Exp: act-author.Name ");
    raise Invalid_Fact_Sequence_For_Activity;
end if;

-- Columns 38 through 62 hold the activity author if it is
-- not null.
if A_Fact(38..41) = "null" then
    -- Do nothing, there was no activity author.
    null;
else
    -- Get the author.
    Activity_Record.Author:= A_Fact(38..62);
    -- Do O(1) procedure call to update the activity in the activity
    -- manager.
    Activity_Manager.Set_Activity_Author
        (Activity_Pointer, Activity_Record.Author);
end if;

Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

-- Note that the last if then construct has already advanced the fact
-- pointer to the next fact. Thus, there is no need to advance the
-- pointer here.
end loop;
end Restore_Activity_Facts;

```

```

-----
-- DATE: 2/19/91 --
-- VERSION: 1.0 --
-- NAME: *** RETRIEVE DATA ELEMENT FACTS *** --
-- MODULE NUMBER: TBD --
-- DESCRIPTION: --
-- When the flag Type_Of_Facts_Flag is set to true, it means the --
-- client procedure wants all the facts that are necessary for --
-- the .esm file. If the flag is false, then the facts for --
-- the expert system are returned. Facts of the same type have --
-- the same format no matter where they are destined. In this --
-- case, the data element name is but a single fact. --
-- data element facts: (retrieved when creating a .esm file or --
-- when performing check syntax) --
-- 1) a predefined attribute name (data-element-name) --
-- 2) the data element name (if the name is null, the word 'null' is --
-- placed in the field. --

```

```

-- ALGORITHM: All simple O(1) statements and 2 O(1) procedure calls. --
-- PASSED VARIABLES: Type_Facts_Flag, Fact_Manager --
-- RETURNS: None --
-- GLOBAL VARIABLES USED: None --
-- GLOBAL VARIABLES CHANGED: None --
-- FILES READ: None --
-- FILES WRITTEN: None --
-- HARDWARE INPUT: None --
-- HARDWARE OUTPUT: None --
-- MODULES CALLED: None --
-- CALLING MODULES: TBD --
-- ORDER-OF: O(1) --
-- AUTHOR(S): Min-fuh Shyong --
-- HISTORY: None (Initial Implementation) --
-----

```

```

procedure Retrieve_Data_Element_Facts
  (Type_Facts_Flag   : in boolean;
   Fact_Manager      : in out
    Environment_Types.Fact_Buffer_Package.Manager_Type) is

```

```

-- Local Declarations --

```

```

Fact_Pointer: Environment_Types.Fact_Buffer_Package.Iterator_Type;
A_Fact: Environment_Types.Fact_String_Type;
Blank_Fact: Environment_Types.Fact_String_Type:= (others => ' ');

```

```

-- Data element declarations --

```

```

Data_Element_Record : Data_Element_Class.Data_Element_Record_Type;
Data_Element_Pointer : Data_Element_Manager.Data_Element_Pointer_Type;

```

```

-- **** varribles for Values(multi-field) *** -----

```

```

  Data_Ele_Values_Iterator:
    Environment_Types.Data_Buffer_Package.Iterator_Type;
  Data_Ele_Values_Line: Environment_Types.DD_Field_Type;

```

```

-- **** variables for description(multi-field) *** ---

```

```

  The_Iterator: Environment_types.Text_Buffer_Package.Iterator_Type;
  A_Description_Line: Environment_Types.DD_Text_Type;

```

```

-- **** variables for Reference(multi-field) ***-----

```

```

  Reference_Iterator :
    Environment_Types.Text_Buffer_Package.Iterator_Type;

```



```

    A_Reference_Line   : Environment_Types.DD_Text_Type;

-- **** variables for changes(multifield) *** --

Version_Iterator      : Environment_Types.Text_Buffer_Package.Iterator_Type;
Version_Line          : Environment_Types.DD_Text_Type;

----- begin -----

begin

-- Clear the passed in fact_manager.  O(1) time.
Environment_Types.Fact_Buffer_Package.Clear(Fact_Manager);

-- Set pointer to beginning of manager.  O(1) time.
Data_Element_Manager.Reset_Data_Element_Iterator;


-- Engage loop to extract the facts associated with an data element. The
-- facts extracted will have the format discussed above.  If there are
-- no data element, this loop won't execute and an empty buffer is the
-- result.
-- This loop runs a times where a is the number of data elements.  At this
-- time there is only one inner loop of O(x) time.  Thus, the time
-- complexity is O(a * x).

while not Data_Element_Manager.Data_Element_Iterator_Done loop
    -- outer loop --
    -- Get a record.  O(1) time.
    Data_Element_Record:= Data_Element_Manager.
                           Value_Of_Data_Element_At_Iterator;

    -- Regardless of the Type_Facts_Flag setting, the data element name is
    -- always added to the fact buffer.

    -- Place the data element name into a fact string at specific positions.
    -- Initialize the fact string to all blanks first.
    -- All string assignments are modeled as O(1).

    -- ****Create Data Element Name Fact**** -----
    A_Fact:= Blank_Fact;
    A_Fact(1..17) := "data-element-name";
    A_Fact(18)    := ' ';

    if Data_Element_Record.Name /= Data_Element_Class.Null_Data_Element_Name then
        A_Fact(19..43) := Data_Element_Record.Name;
    else
        A_Fact(19..22) := "null";

```

```

end if;
-- Store this fact in the fact buffer.  O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
  (A_Fact, Fact_Manager, Fact_Pointer);
                                --(data-element-name Name)
                                -- (data-element-name null)

-- ****Create Data Element Data_Type Fact**** -----
A_Fact:= Blank_Fact;
A_Fact(1..17) := "data-element-type";
A_Fact(18)    := ' ';
A_Fact(19..43) := Data_Element_Record.Name;
A_Fact(44)    := ' ';

-- This if then construct determines what goes into the last field.
-- If the data element data_type is not null then create a fact with the
-- data_type in it.  Flag setting doesn't matter here.
if Data_Element_Record.Data_Type /=
    Data_Element_Class.Null_Data_Element_Data_Type then
  -- All statements modeled as O(1) time.
  A_Fact(45..69) := Data_Element_Record.Data_Type;
else
  -- The activity number is null, so create a null fact for
  -- either the .esm file or the expert system.  Again, the flag
  -- setting does not matter.  All O(1) time.
  A_Fact(45..48) := "null";
end if;

-- Store this fact in the fact buffer.  O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
  (A_Fact, Fact_Manager, Fact_Pointer);
                                -- (data-element-type Name Data-Type)
                                -- (data-element-type Name null)

-- ****Create Data Element minimum Fact**** -----
A_Fact:= Blank_Fact;
A_Fact(1..20) := "data-element-minimum";
A_Fact(21)    := ' ';
A_Fact(22..46) := Data_Element_Record.Name;
A_Fact(47)    := ' ';

if Data_Element_Record.Minimum /=
    Data_Element_Class.Null_Data_Element_Value then
  A_Fact(48..62) := Data_Element_Record.Minimum;
else
  A_Fact(48..51) := "null";
end if;

```

```

-- Store this fact in the fact buffer.  O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
  (A_Fact, Fact_Manager, Fact_Pointer);
                                --(data-element-minimum Name Minimum)
                                -- (data-element-minimum Name null)

-- ****Create Data Element Maximum Fact**** -----
A_Fact:= Blank_Fact;
A_Fact(1..20) := "data-element-maximum";
A_Fact(21)    := ' ';
A_Fact(22..46) := Data_Element_Record.Name;
A_Fact(47)    := ' ';

-- This if then construct determines what goes into the last field.
-- If the data element maximum is not null then create a fact with the
-- maximum in it.  Flag setting doesn't matter here.

if Data_Element_Record.Maximum /=
    Data_Element_Class.Null_Data_Element_Value then
  -- All statements modeled as O(1) time.
  A_Fact(48..62) := Data_Element_Record.Maximum;

else
  -- The data element maximum is null, so create a null fact for
  -- either the .esm file or the expert system.  Again, the flag
  -- setting does not matter.  All O(1) time.
  A_Fact(48..51) := "null";
end if;

-- Store this fact in the fact buffer.  O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
  (A_Fact, Fact_Manager, Fact_Pointer);
                                -- (data-element-maximum Name Maximum)
                                -- (data-element-maximum Name null)

-- ****Create Data Element data range Fact**** -----
A_Fact:= Blank_Fact;
A_Fact(1..23) := "data-element-data-range";
A_Fact(24)    := ' ';
A_Fact(25..49) := Data_Element_Record.Name;
A_Fact(50)    := ' ';

-- This if then construct determines what goes into the last field.
-- If the data element data range is not null then create a fact with the
-- range in it.  Flag setting doesn't matter here.

if Data_Element_Record.Data_Range /= Data_Element_Class.Null_Data_Element_Value then
  -- All statements modeled as O(1) time.

```

```

    A_Fact(51..65) := Data_Element_Record.Data_Range;

else
    -- The data element data range is null, so create a null fact for
    -- either the .esm file or the expert system. Again, the flag
    -- setting does not matter. All O(1) time.
    A_Fact(51..54) := "null";
end if;

-- Store this fact in the fact buffer. O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

                                -- (data-element-data-range Name Data_Range)
                                -- (data-element-data-range Name null)

-- ****Create one or more data element values  Facts****-----

-- If the data element values is null then only a single fact is
-- created regardless of the flag setting.
if Environment_Types.Data_Buffer_Package.Is_Empty
    (Data_Element_Record.Values) then

    -- Create a null fact.
    A_Fact:= Blank_Fact;
    A_Fact(1..19) := "data-element-values";
    A_Fact(20)    := ' ';
    A_Fact(21..45) := Data_Element_Record.Name;
    A_Fact(46)    := ' ';
    A_Fact(47..50):= "null";

    -- Store this fact in the fact buffer. O(1) time.
    Environment_Types.Fact_Buffer_Package.Add_Item
        (A_Fact, Fact_Manager, Fact_Pointer);

-- A false flag setting means the fact is for the expert system.
-- For a value, we don't want the whole value in the
-- working memory, so just store a "not-null" string'

elsif Type_Facts_Flag = False then

    -- Create a fact that shows the description is not null.

```

```

A_Fact:= Blank_Fact;
A_Fact(1..19) := "data-element-values";
A_Fact(20)    := ' ';
A_Fact(21..45) := Data_Element_Record.Name;
A_Fact(46)    := ' ';
A_Fact(47..54) := "not-null";

-- Store this fact in the fact buffer.  O(1) time.

Environment_Types.Fact_Buffer_Package.Add_Item
  (A_Fact, Fact_Manager, Fact_Pointer);

else
  -- At this point we know the flag is true which means the fact
  -- is to go to the .esm file.  However, there may be multiple lines
  -- in the values thus a loop is required.

  -- Set iterator to first line of description.
  Environment_Types.Data_Buffer_Package.Initialize_Iterator
    (Data_Ele_Values_Iterator, Data_Element_Record.Values);

  -- Engage loop to get each line of the values and
  -- make it a fact.  This loop is O(x) time where x is the
  -- number of lines in the description.
  while not Environment_Types.Data_Buffer_Package.Is_Done
    (Data_Ele_Values_Iterator) loop

    -- Retrieve a single line of text.  O(1) time.
    Data_Ele_Values_Line:= Environment_Types.Data_Buffer_Package.
      Value_Of_Item(Data_Ele_Values_Iterator);

    -- Create a fact representing a single line of the description.
    A_Fact:= Blank_Fact;
    A_Fact(1..19) := "data-element-values";
    A_Fact(20)    := ' ';
    A_Fact(21..45) := Data_Element_Record.Name;
    A_Fact(46)    := ' ';
    A_Fact(47..71) := Data_Ele_Values_Line;

    -- Store this fact in the fact buffer.  O(1) time.
    Environment_Types.Fact_Buffer_Package.Add_Item
      (A_Fact, Fact_Manager, Fact_Pointer);

    -- Advance pointer by one to next line.

    Environment_Types.Data_Buffer_Package.
      Get_Next(Data_Ele_Values_Iterator);
  end loop;
end if;

-- (data-element-values Name null)
-- (data-element-values Name Line1) ...

```

```
-- ****Create one or more Data Element Description Facts****-----
```

```
-----
-- If the data element description is null then only a single fact is
-- created regardless of the flag setting.
```

```
if Environment_Types.Text_Buffer_Package.Is_Empty
(Data_Element_Record.Description) then
```

```
-- Create a null fact.
```

```
A_Fact:= Blank_Fact;
```

```
A_Fact(1..9) := "data-desc";
```

```
A_Fact(10) := ' ';
```

```
A_Fact(11..35) := Data_Element_Record.Name;
```

```
A_Fact(36) := ' ';
```

```
A_Fact(37..40):= "null";
```

```
-- Store this fact in the fact buffer. O(1) time.
```

```
Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);
```

```
-- A false flag setting means the fact is for the expert system.
```

```
-- For a description, we don't want the whole description in the
```

```
-- working memory, so just store a "not-null" string!
```

```
elsif Type_Facts_Flag = False then
```

```
-- Create a fact that shows the description is not null.
```

```
A_Fact:= Blank_Fact;
```

```
A_Fact(1..9) := "data-desc";
```

```
A_Fact(10) := ' ';
```

```
A_Fact(11..35) := Data_Element_Record.Name;
```

```
A_Fact(36) := ' ';
```

```
A_Fact(37..44):= "not-null";
```

```
-- Store this fact in the fact buffer. O(1) time.
```

```
Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);
```

```
else
```

```
-- At this point we know the flag is true which means the fact
```

```
-- is to go to the .esm file. However, there may be multiple lines
```

```
-- in the description thus a loop is required.
```

```
-- Set iterator to first line of description.
```

```
Environment_Types.Text_Buffer_Package.Initialize_Iterator
(The_Iterator, Data_Element_Record.Description);
```

```
-- Engage loop to get each line of the description and
```

```
-- make it a fact. This loop is O(x) time where x is the
```

```
-- number of lines in the description.
```

```
while not Environment_Types.Text_Buffer_Package.Is_Done
```

```

(The_Iterator) loop

-- Retrieve a single line of text.  O(1) time.
A_Description_Line:= Environment_Types.Text_Buffer_Package.
    Value_Of_Item(The_Iterator);

-- Create a fact representing a single line of the description.
A_Fact:= Blank_Fact;
A_Fact(1..9) := "data-desc";
A_Fact(10)   := ' ';
A_Fact(11..35) := Data_Element_Record.Name;
A_Fact(36)   := ' ';
A_Fact(37..96) := A_Description_Line;

-- Store this fact in the fact buffer.  O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

-- Advance pointer by one to next line.
Environment_Types.Text_Buffer_Package.Get_Next(The_Iterator);
end loop;
end if;

-- (data-desc Name null)
-- (data-desc Name not-null)
-- (data-desc Name Description_Line1)

-- ****Create one or more data reference Facts****-----
-----
-- If the data reference is null then only a single fact is
-- created regardless of the flag setting.
if Environment_Types.Text_Buffer_Package.Is_Empty
    (Data_Element_Record.Reference) then

    -- Create a null fact.
    A_Fact:= Blank_Fact;
    A_Fact(1..8) := "data-ref";
    A_Fact(9)   := ' ';
    A_Fact(10..34) := Data_Element_Record.Name;
    A_Fact(35)   := ' ';
    A_Fact(36..39) := "null";

    -- Store this fact in the fact buffer.  O(1) time.
    Environment_Types.Fact_Buffer_Package.Add_Item
        (A_Fact, Fact_Manager, Fact_Pointer);

-- A false flag setting means the fact is for the expert system.
-- For a reference, we don't want the whole reference in the
-- working memory, so just store a "not-null" string'

```

```

elsif Type_Facts_Flag = False then

    -- Create a fact that shows the reference is not null.
    A_Fact:= Blank_Fact;
    A_Fact(1..8) := "data-ref";
    A_Fact(9)    := ' ';
    A_Fact(10..34) := Data_Element_Record.Name;
    A_Fact(35)   := ' ';
    A_Fact(36..43) := "not-null";

    -- Store this fact in the fact buffer. O(1) time.
    Environment_Types.Fact_Buffer_Package.Add_Item
        (A_Fact, Fact_Manager, Fact_Pointer);

else

    -- At this point we know the flag is true which means the fact
    -- is to go to the .esm file. However, there may be multiple lines
    -- in the reference thus a loop is required.

    -- Set iterator to first line of description.
    Environment_Types.Text_Buffer_Package.Initialize_Iterator
        (Reference_Iterator, Data_Element_Record.Reference);

    -- Engage loop to get each line of the reference and
    -- make it a fact. This loop is O(x) time where x is the
    -- number of lines in the reference.
    while not Environment_Types.Text_Buffer_Package.Is_Done
        (Reference_Iterator) loop

        -- Retrieve a single line of text. O(1) time.
        A_Reference_Line:= Environment_Types.Text_Buffer_Package.
            Value_Of_Item(Reference_Iterator);

        -- Create a fact representing a single line of the description.
        A_Fact:= Blank_Fact;
        A_Fact(1..8) := "data-ref";
        A_Fact(9)    := ' ';
        A_Fact(10..34) := Data_Element_Record.Name;
        A_Fact(35)   := ' ';
        A_Fact(36..95) := A_Reference_Line;

        -- Store this fact in the fact buffer. O(1) time.
        Environment_Types.Fact_Buffer_Package.Add_Item
            (A_Fact, Fact_Manager, Fact_Pointer);

        -- Advance pointer by one to next line.
        Environment_Types.Text_Buffer_Package.Get_Next(Reference_Iterator);
    end loop;
end if;

-- (data-ref Name not-null)

```



```

-- (data-ref Name null)
-- (data-ref Name Reference_Line1)

-- ***** Create data reference type facts 120390 *****

A_Fact      := Blank_Fact;
A_Fact(1..13) := "data-ref-type";
A_Fact(14)   := ' ';
A_Fact(15..39) := Data_Element_Record.Name;
A_Fact(40)   := ' ';

if Data_Element_Record.Reference_Type /=
    Environment_Types.Null_Reference_Type then
    A_Fact(41..65) := Data_Element_Record.Reference_Type;
else
    A_Fact(41..44) := "null";
end if;

Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

-- (data-ref-type Name Reference_Type) or
-- (data-ref-type Name null)

-- ***** Create data element Version facts *****

A_Fact      := Blank_Fact;
A_Fact(1..12) := "data-ele-ver";
A_Fact(13)   := ' ';
A_Fact(14..38) := Data_Element_Record.Name;
A_Fact(39)   := ' ';

if Data_Element_Record.Version /=
    Data_Element_Class.Null_Data_Element_Version_number then
    A_Fact(40..49) := Data_Element_Record.Version;
else
    A_Fact(40..43) := "null";
end if;

Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

-- (data-ele-ver Name Data-Element-Version) or
-- (data-ele-ver Name null)

-- *** Create one or more data element Version-Changes facts -----

```

```

-- if the data element version changes is null then only a single
-- fact is created regardless of the flag setting

if Environment_Types.Text_Buffer_package.Is_Empty
    (Data_Element_Record.Version_Changes) then
    -- true it is empty

        A_Fact          := Blank_Fact;
        A_Fact(1..12)   := "data-e-v-chg";
        A_Fact(13)      := ' ';
        A_Fact(14..38)  := Data_Element_Record.Name;
        A_Fact(39)      := ' ';
        A_Fact(40..43)  := "null";

Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

elseif Type_Facts_Flag = false then

    A_Fact    := Blank_Fact;
    A_Fact(1..12) := "data-e-v-chg";
    A_Fact(13) := ' ';
    A_Fact(14..38) := Data_Element_Record.Name;
    A_Fact(39) := ' ';
    A_Fact(40..47) := "not-null";

Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

else
    -- version not empty show the version changes to .esm --
Environment_Types.Text_Buffer_Package.Initialize_Iterator
    (Version_Iterator, Data_Element_Record.Version_Changes);

    -- Engage a loop to get  each version of changes  and make it a fact

while not Environment_Types.Text_Buffer_Package.Is_Done
    (Version_Iterator) loop

Version_Line :=
    Environment_Types.Text_Buffer_Package.Value_Of_Item(Version_Iterator);

    A_Fact    := Blank_Fact;
    A_Fact(1..12) := "data-e-v-chg";
    A_Fact(13) := ' ';
    A_Fact(14..38) := Data_Element_Record.Name;
    A_Fact(39) := ' ';

```

```

        A_Fact(40..99) := Version_Line;    -- (data-e-v-chg Name null)
                                           -- (data-e-v-chg Name not-null)
                                           -- (data-e-v-chg Name Version-changes)
Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

Environment_Types.Text_Buffer_Package.Get_Next(Version_Iterator);

end loop;
end if;

-- ****Create data element Date Fact ****-----
A_Fact:= Blank_Fact;
A_Fact(1..13) := "data-ele-date";
A_Fact(14)    := ' ';
A_Fact(15..39) := Data_Element_Record.Name;
A_Fact(40)    := ' ';

if Data_Element_Record.Date /= Environment_Types.Null_Date then
A_Fact(41..48) := Data_Element_Record.Date;

else

    A_Fact(41..44) := "null";
end if;

-- Store this date fact in the fact buffer.  O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
(A_Fact, Fact_Manager, Fact_Pointer);

-- (date-ele-date Name mm/dd/yy)
-- (data-ele-date Name null)

-- ****Create data element Author Fact****-----
A_Fact:= Blank_Fact;
A_Fact(1..15) := "data-ele-author";
A_Fact(16)    := ' ';
A_Fact(17..41) := Data_Element_Record.Name;
A_Fact(42)    := ' ';

-- This if then construct determines what goes into the last field.
-- If the data element author is not null then create a fact with the
-- data element author in it.  Flag setting doesn't matter here.

if Data_Element_Record.Author /= Environment_Types.Null_Author_Name then
-- All statements modeled as O(1) time.
A_Fact(43..67) := Data_Element_Record.Author;
else
-- The data element author is null, so create a null fact for

```

```

-- either the .esm file or the expert system. Again, the flag
-- setting does not matter. All O(1) time.
  A_Fact(43..46) := "null";
end if;

-- Store this fact in the fact buffer. O(1) time.
Environment_Types.Fact_Buffer_Package.Add_Item
  (A_Fact, Fact_Manager, Fact_Pointer);

-- (data-ele-author Name Author)
-- (data-ele-author Name null)

Data_Element_Manager.Advance_Iterator_To_Next_Data_Element;

end loop;
end Retrieve_Data_Element_Facts;

-----
-- DATE: 2/21/91 --
-- VERSION: 1.0 --
-- NAME: ***PROCEDURE RESTORE DATA ELEMENT FACTS*** --
-- MODULE NUMBER: TBD --
-- DESCRIPTION: This procedure accepts a buffer of data element facts --
-- and restores that information into the data element manager. Of --
-- special note is that the procedure assumes the facts are in the --
-- same order in which they were stored. --
-- ALGORITHM: A single while loop controls the execution with an --
-- embedded call to an O(i) procedure. --
-- PASSED VARIABLES: The_Fact_Buffer (contains the facts) --
-- RETURNS: None --
-- GLOBAL VARIABLES USED: None --
-- GLOBAL VARIABLES CHANGED: None --
-- FILES READ: None --
-- FILES WRITTEN: None --
-- HARDWARE INPUT: None --
-- HARDWARE OUTPUT: None --
-- MODULES CALLED: None --
-- CALLING MODULES: Essential_IO.Restore_Project --
-- ORDER-OF: order of is  $O(a * \max(x, z * (a * z)))$  where a is the --
-- number of data elements, x is the number of lines in a description --
-- and z is the number of reference that a data element has. Note --
-- that this order of may change when more of the data element --
-- manager facts are restored. --
-- Note that all string slice operations are modeled as O(1) time. --
-- AUTHOR(S): Min-fuh Shyong --
-- HISTORY: none (Initial Implementation) --

```

```

-----
procedure Restore_Data_Element_Facts
  (The_Fact_Buffer : in
   Environment_Types.Fact_Buffer_Package.Manager_Type) is

  -- Local Declarations --

  Fact_Pointer: Environment_Types.Fact_Buffer_Package.Iterator_Type;
  A_Fact: Environment_Types.Fact_String_Type;
  First_Char: natural:= 0;
  Char_Position: natural:= 0;
  Temp_Pos: natural:= 0;
  More_Descriptions_Flag: boolean;

  -- Data Element Related Declarations --

  Data_Element_Record: Data_Element_Class.Data_Element_Record_Type;
  Data_Element_Pointer: Data_Element_Manager.Data_Element_Pointer_Type;
  Null_Data_Element_Record: Data_Element_Class.Data_Element_Record_Type;

  The_Iterator: Environment_Types.Text_Buffer_Package.Iterator_Type;

  Data_Ele_Values_Line : Environment_Types.DD_Field_Type;
  A_Description_Line   : Environment_Types.DD_Text_Type;
  A_Reference_Line     : Environment_Types.DD_Text_Type;
  Version_Line        : Environment_Types.DD_Text_Type;

  Found_Flag: boolean:= False;
  Result_Flag: boolean;

  -- Exception --
  -- This exception is declared here because the Essential IO package does
  -- not check to see the facts are in any specific order.

  Invalid_Fact_Sequence_For_Data_Element: exception;
  Data_Element_Hierarchy_Error_During_Restore: exception;

  -- ----- begin Restore Data Element -----
begin
  -- Check for empty buffer of facts.  If empty, do nothing.
  if Environment_Types.Fact_Buffer_Package.Is_Empty(The_Fact_Buffer) then
    return;
  end if;

  -- Initialize iterator to first fact.
  Environment_Types.Fact_Buffer_Package.Initialize_Iterator
    (Fact_Pointer, The_Fact_Buffer);

  -- Engage loop to extract the data element facts from a buffer

```

```

-- one at a time. This loop will execute a times -- once for
-- each data element. Note that there are many facts associated with
-- a single data element. This loop runs a times. The loop has one
-- inner loop of order x and one procedure call of (a * z). Thus,
-- order of is  $O(a * \max(x, z * (a * z)))$ 

```

```

While not Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) loop

```

```

-- Get a record.

```

```

A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);

```

```

-- Since we put the information in the string, we know the
-- exact columns where information should be.
-- All string assignments are modeled as  $O(1)$ ;
-- Insure the fields are all blanks.

```

```

Data_Element_Record:= Null_Data_Element_Record;

```

```

-- *** Restore Data Element Name *** -----

```

```

-- The first fact should be the name.

```

```

if A_Fact(1..17) /= "data-element-name" then

```

```

    Text_IO.Put_Line(A_Fact);

```

```

    Text_IO.Put_Line("I am here: data-element-name. ");

```

```

    raise Invalid_Fact_Sequence_For_Data_Element;

```

```

end if;

```

```

-- The Data Element name must be in columns 19 through 43.

```

```

Data_Element_Record.Name:= A_Fact(19..43);

```

```

-- Check to see if Data Element already exists.  $O(a)$  call.

```

```

Data_Element_Manager.Data_Element_Exists(Data_Element_Record.Name,
        Data_Element_Pointer, Found_Flag);

```

```

if Found_Flag = False then

```

```

    -- Do  $O(a * z)$  procedure call to create a data element.

```

```

    Data_Element_Manager.Create_Data_Element

```

```

        (Data_Element_Record.Name, Data_Element_Pointer);

```

```

end if;

```

```

-- Advance pointer to next fact in manager.  $O(1)$ .

```

```

Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

```

```

-- Get a fact.  $O(1)$  time.

```

```

A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);

```

```

-- *** Restore Data Element Data Type *** -----

```

```

-- This fact should be the Data Element Data Type.

```

```

if A_Fact(1..17) /= "data-element-type" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-element-type.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 19 through 43 must be the same data element name.
if A_Fact(19..43) /= Data_Element_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-element-type.Name.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 45 through 69 hold the data element data type if it is
-- not null.
if A_Fact(45..48) = "null" then
    -- Do nothing, there was no data element data type.
    null;
else
    -- Get the number.
    Data_Element_Record.Data_Type:= A_Fact(45..69);
    -- Do O(1) procedure call to update the data type in the data element
    -- manager.

    Data_Element_Manager.Set_Data_Element_Data_Type
        (Data_Element_Pointer, Data_Element_Record.Data_Type);
end if;

-- Advance pointer to next fact in manager.
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
-- If the fact buffer is empty at this point there is an error
-- in the format.
if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-element-type.Is_Done  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Get a fact.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);

-- *** Restore Data Element Minimum *** -----

-- This fact should be the Data Element Minimum.

if A_Fact(1..20) /= "data-element-minimum" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-element-minimum.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

```

```

-- Columns 22 through 46 must be the same data element name.
if A_Fact(22..46) /= Data_Element_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-element-minimum.Name.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 48 through 62 hold the data element minimum if it is
-- not null.
if A_Fact(48..51) = "null" then
    -- Do nothing, there was no data element minimum.
    null;
else
    -- Get the minimum.
    Data_Element_Record.Minimum:= A_Fact(48..62);
    -- Do O(1) procedure call to update the minimum in the data element
    -- manager.

    Data_Element_Manager.Set_Data_Element_Minimum
        (Data_Element_Pointer, Data_Element_Record.Minimum);
end if;

-- Advance pointer to next fact in manager.
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
-- If the fact buffer is empty at this point there is an error
-- in the format.
if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-element-minimum.Is_Done.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Get a fact.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

-- *** Restore Data Element Maximum *** -----

-- This fact should be the Data Element Maximum.

if A_Fact(1..20) /= "data-element-maximum" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-element-maximum  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 22 through 46 must be the same data element name.
if A_Fact(22..46) /= Data_Element_Record.Name then

```



```

        Text_IO.Put_Line(A_Fact);
        Text_IO.Put_Line("I am Exp:  data-element-maximum.Name.  ");
        raise Invalid_Fact_Sequence_For_Data_Element;
    end if;

-- Columns 48 through 62 hold the data element maximum if it is
-- not null.
if A_Fact(48..51) = "null" then
    -- Do nothing, there was no data element maximum.
    null;
else
    -- Get the maximum.
    Data_Element_Record.Maximum:= A_Fact(48..62);
    -- Do O(1) procedure call to update the maximum in the data element
    -- manager.

    Data_Element_Manager.Set_Data_Element_Maximum
        (Data_Element_Pointer, Data_Element_Record.Maximum);
end if;

-- Advance pointer to next fact in manager.
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
-- If the fact buffer is empty at this point there is an error
-- in the format.
if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp:  data-element-maximum.Is_done.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Get a fact.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);

-- *** Restore Data Element Data Range *** -----

-- This fact should be the Data Element Data range.

if A_Fact(1..23) /= "data-element-data-range" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp:  data-element-data-range.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 25 through 49 must be the same data element name.
if A_Fact(25..49) /= Data_Element_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp:  data-element-data-range.Name  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

```

```

-- Columns 51 through 65 hold the data element data range if it is
-- not null.
if A_Fact(51..54) = "null" then
    -- Do nothing, there was no data element data range.
    null;
else
    -- Get the range.

    Data_Element_Record.Data_Range:= A_Fact(51..65);
    -- Do O(1) procedure call to update the data range in the data element
    -- manager.

    Data_Element_Manager.Set_Data_Element_Data_Range
        (Data_Element_Pointer, Data_Element_Record.Data_Range);
end if;

-- Advance pointer to next fact in manager. ---

Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
-- If the fact buffer is empty at this point there is an error
-- in the format.
if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-element-data-range.Is_Done.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Get a fact.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);

-- *** Restore Data Element Values facts *** -----
if A_Fact(1..19) /= "data-element-values" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-element-values.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 21 through 45 must be the same data element name.
if A_Fact(21..45) /= Data_Element_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-element-values.Name.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- If the values list is null then we are done with this attribute.
-- Need only to advance the pointer by one for the outer loop.

```

```

if A_Fact(47..50) = "null" then
    -- There is no values list for the data element, so just advance
    -- the fact pointer.
    -- Advance pointer to next fact in manager. O(1) time.
    Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
else
    -- There must be one or more values.
    -- This loop will run z times where z is the number of values.

while A_Fact(1..19) = "data-element-values" loop

    -- Columns 21 through 45 must be the same data element name.
    -- O(1) time complexity.
    if A_Fact(21..45) /= Data_Element_Record.Name then
        Text_IO.Put_Line(A_Fact);
        Text_IO.Put_Line("I am Exp: data-element-values.Name. in while  ");
        raise Invalid_Fact_Sequence_For_Data_Element;
    end if;

    -- Pull a Value from the fact.
    Data_Ele_Values_Line:= A_Fact(47..71);

    -- In order to add a value, the Data Element Manager
    -- requires that the value already exist as a data element.
    -- Thus, must create the data element first if needed.

    -- Check to see if data element already exists. O(a) call.
    Data_Element_Manager.Data_Element_Exists(Data_Ele_Values_Line,
        Data_Element_Pointer, Found_Flag);
    if Found_Flag = False then
        -- Do O(a * z) procedure call to create a data element.
        Data_Element_Manager.Create_Data_Element
            (Data_Ele_Values_Line, Data_Element_Pointer);
    end if;

    -- Do another O(a * z) procedure call to add this data element

Data_Element_Manager.Set_Data_Element_Values
    (Data_Element_Pointer, Data_Element_Record.Values);

    -- Must now call Data Element Exists again in order to reset the
    -- pointer for any future operations. O(a) time.
    Data_Element_Manager.Data_Element_Exists(Data_Element_Record.Name,
        Data_Element_Pointer, Found_Flag);

```

```

-- Advance pointer to next fact in manager.
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

-- If it is not empty get the next fact. O(1) time.
if not Environment_Types.Fact_Buffer_Package.Is_Done
  (Fact_Pointer) then
  A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);
else
  Text_IO.Put_Line(A_Fact);
  Text_IO.Put_Line("I am Exp: data-element-values in else  ");
  raise Invalid_Fact_Sequence_For_Data_Element;
end if;
end loop;
end if;

-- *** Restore Data Element Description Facts *** -----

if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
  Text_IO.Put_Line(A_Fact);
  Text_IO.Put_Line("I am Exception: data-element Is_Done  ");
  raise Invalid_Fact_Sequence_For_Data_Element;
end if;

A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
  (Fact_Pointer);

-- The series of fact(s) should be the data element description.
-- There is at least one data-desc fact and possible more.
if A_Fact(1..9) /= "data-desc" then
  Text_IO.Put_Line(A_Fact);
  Text_IO.Put_Line("I am Exp: data-desc.  ");
  raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 11 through 35 must be the same data element name.
if A_Fact(11..35) /= Data_Element_Record.Name then
  Text_IO.Put_Line(A_Fact);
  Text_IO.Put_Line("I am Exp: data-desc.Name  ");
  raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- If the description is null then we are done with this attribute.
-- Need only to advance the pointer by one for the outer loop.
if A_Fact(37..40) = "null" then
  -- There is no description for the data element, so just advance
  -- the fact pointer.
  -- Advance pointer to next fact in manager. O(1) time.
  Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

```

```

else
  -- There must be one or more lines in the description.
  -- This loop will run x times where x is the number of lines in the
  -- description.
  while A_Fact(1..9) = "data-desc" loop
    -- I realize this check is repetitive on the first iteration.
    -- Columns 11 through 35 must be the same data element name.
    -- O(1) time complexity.
    if A_Fact(11..35) /= Data_Element_Record.Name then
      Text_IO.Put_Line(A_Fact);
      Text_IO.Put_Line("I am Exp: data-desc.Name in while  ");
      raise Invalid_Fact_Sequence_For_Data_Element;
    end if;

    -- Pull the description from the fact.
    A_Description_Line:= A_Fact(37..96);

    -- Add the description to the description part of the
    -- data element record. O(1) time.

    Environment_Types.Text_Buffer_Package.Add_Item
      (A_Description_Line, Data_Element_Record.Description, The_Iterator);

    -- Advance pointer to next fact in manager.
    Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

    -- If it is not empty get the next fact. O(1) time.
    if not Environment_Types.Fact_Buffer_Package.Is_Done
      (Fact_Pointer) then
      A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);
    else
      -- If this is the last fact of the last data element exit the
      -- loop.
      Text_IO.Put_Line(A_Fact);
      Text_IO.Put_Line("I am Exp: data-desc in else  ");
      raise Invalid_Fact_Sequence_For_Data_Element;
    end if;
  end loop;

  -- There were one or more lines in the description so now must
  -- place them with the data element in the data element manager. O(1).
  Data_Element_Manager.Set_Data_Element_Description
    (Data_Element_Pointer, Data_Element_Record.Description);

end if;

-- If the fact buffer is empty at this point there is an error
-- in the format. O(1) time. I know this because
-- Retrieve_Data_Element_Facts will at least put a null entry

```

```

if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-desc.Is_Done  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Get a fact.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

-- The series of fact(s) should be the data element description.
-- There is at least one data-desc fact and possible more.

-- *** Restore Data Element Reference Facts *** -----
if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-ref.Is_Done.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;
-- Get a fact.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

-- The series of fact(s) should be the Data Element Reference.
-- There is at least one data-ref fact and possible more.
if A_Fact(1..8) /= "data-ref" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-ref.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 10 through 34 must be the same data element name.
if A_Fact(10..34) /= Data_Element_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-ref.Name  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- If the Reference is null then we are done with this attribute.
-- Need only to advance the pointer by one for the outer loop.

if A_Fact(36..39) = "null" then
    -- There is no reference for the data element, so just advance
    -- the fact pointer.
    -- Advance pointer to next fact in manager. O(1) time.

```

```

Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

else
  -- There must be one or more lines in the reference.
  -- This loop will run x times where x is the number of lines in the
  -- reference.
  while A_Fact(1..8) = "data-ref" loop
    -- I realize this check is repetitive on the first iteration.
    -- Columns 10 through 34 must be the same data element name.
    -- O(1) time complexity.
    if A_Fact(10..34) /= Data_Element_Record.Name then
      Text_IO.Put_Line(A_Fact);
      Text_IO.Put_Line("I am Exp: data-ref.Name in while  ");
      raise Invalid_Fact_Sequence_For_Data_Element;
    end if;

    -- Pull the reference from the fact.
    A_Reference_Line:= A_Fact(36..95);

    -- Add the reference to the reference part of the
    -- data element record. O(1) time.
    Environment_Types.Text_Buffer_Package.Add_Item
      (A_Reference_Line, Data_Element_Record.Reference, The_Iterator);

    -- Advance pointer to next fact in manager.
    Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

    -- If it is not empty get the next fact. O(1) time.

    if not Environment_Types.Fact_Buffer_Package.Is_Done
      (Fact_Pointer) then

      A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);
    else
      -- If this is the last fact of the last data element exit the
      -- loop.
      Text_IO.Put_Line(A_Fact);
      Text_IO.Put_Line("I am Exp: data-ref. in else  ");
      raise Invalid_Fact_Sequence_For_Data_Element;
    end if;
  end loop;

  -- There were one or more lines in the reference so now must
  -- place them with the data element in the data element manager. O(1).
  Data_Element_Manager.Set_Data_Element_Reference
    (Data_Element_Pointer, Data_Element_Record.Reference);

end if;

```

```

-- *** Restore Data Element Reference Type Facts *** -----
if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exception: data-element reference type Is_Done  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Advance pointer to next fact in manager. O(1).

--Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

-- Get a fact. O(1) time.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

-- This fact should be the Data Element Reference Type.
if A_Fact(1..13) /= "data-ref-type" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-ref-type.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 15 through 39 must be the same data element name.
if A_Fact(15..39) /= Data_Element_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-ref-type.Name  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 41 through 65 hold the Data Element Reference Type if it is
-- not null.

if A_Fact(41..44) = "null" then
    -- Do nothing, there was no Data Element Reference Type.
    null;
else
    -- Get the Reference type

    Data_Element_Record.Reference_Type:= A_Fact(41..65);
    -- Do O(1) procedure call to update the data element in the
    -- data element manager.

    Data_Element_Manager.Set_Data_Element_Reference_Type
        (Data_Element_Pointer, Data_Element_Record.Reference_Type);
end if;

```



```

-- *** get Data Element Version Facts *** -----

if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exception: data-   ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

    -- Advance pointer to next fact in manager. O(1).
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

-- Get a fact. O(1) time.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

-- This fact should be the Data Element Version.

if A_Fact(1..12) /= "data-ele-ver" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-ele-ver   ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 14 through 38 must be the same Data element name.
if A_Fact(14..38) /= Data_Element_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-ele-ver.Name   ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 40 through 49 hold the data element version if it is
-- not null.

if A_Fact(40..43) = "null" then
    -- Do nothing, there was no data element version.
    null;
else
    -- Get the version.
    Data_Element_Record.version:= A_Fact(40..49);
    -- Do O(1) procedure call to update the data element in the
    -- data element manager.
    Data_Element_Manager.Set_Data_Element_Version
        (Data_Element_Pointer, Data_Element_Record.Version);
end if;

-- *** get Data Element Version Changes Facts *** -----

-- Advance pointer to next fact in manager.

```

```

Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
-- If the fact buffer is empty at this point there is an error
-- in the format.

if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-e-v-chg.Is_Done  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Get a fact.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);

-- The series of fact(s) should be the Data Element Version Changes.
-- There is at least one data-e-v-chg fact and possible more.
if A_Fact(1..12) /= "data-e-v-chg" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-e-v-chg.  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 14 through 38 must be the same data element version name.
if A_Fact(14..38) /= Data_Element_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-e-v-chg.Name  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- If the version change is null then we are done with this attribute.
-- Need only to advance the pointer by one for the outer loop.

if A_Fact(40..43) = "null" then
    -- There is no version change for the data element, so just advance
    -- the fact pointer.
    -- Advance pointer to next fact in manager.  O(1) time.

    Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

else
    -- There must be one or more lines in the version changes.
    -- This loop will run x times where x is the number of times in the
    -- version change.

    while A_Fact(1..12) = "data-e-v-chg" loop
        -- I realize this check is repetitive on the first iteration.

```

```

-- Columns 14 through 38 must be the same data element name.
-- O(1) time complexity.
if A_Fact(14..38) /= Data_Element_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-e-v-chg. while  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Pull the version from the fact.
Version_Line:= A_Fact(40..99);

-- Add the version change to the Version Changes part of the
-- Data Element record. O(1) time.

Environment_Types.Text_Buffer_Package.Add_Item
    (Version_Line, Data_Element_Record.Version_Changes, The_Iterator);

-- Advance pointer to next fact in manager.

Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

-- If it is not empty get the next fact. O(1) time.
if not Environment_Types.Fact_Buffer_Package.Is_Done
    (Fact_Pointer) then
    A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);
else
    -- If this is the last fact of the last data element exit the
    -- loop.
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-e-v-chg. else  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;
end loop;

-- There were one or more lines in the version change so now must
-- place them with the data element in the data element manager. O(1).
Data_Element_Manager.Set_Data_Element_Version_Comments
    (Data_Element_Pointer, Data_Element_Record.Version_Changes);

end if;

--*** get Data Element Date Facts *** -----

if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exception: data-  ");
    raise Invalid_Fact_Sequence_For_Data_Element;

```

```

end if;

-- Get a fact. O(1) time.
A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
        (Fact_Pointer);

-- This fact should be the data element date.

if A_Fact(1..13) /= "data-ele-date" then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-ele-date  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 15 through 39 must be the same Data element name.
if A_Fact(15..39) /= Data_Element_Record.Name then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exp: data-ele-data.Name  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Columns 41 through 48 hold the data element date if it is
-- not null.
if A_Fact(41..44) = "null" then
    -- Do nothing, there was no data element date.
    null;
else
    -- Get the date.

    Data_Element_Record.Date:= A_Fact(41..48);
    -- Do O(1) procedure call to update the Data element in the
    -- Data Element manager.

    Data_Element_Manager.Set_Data_Element_Date
        (Data_Element_Pointer, Data_Element_Record.Date);
end if;

-- *** Get Data Element Author Facts *** -----

if Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) then
    Text_IO.Put_Line(A_Fact);
    Text_IO.Put_Line("I am Exception: data-element-author Is_Done  ");
    raise Invalid_Fact_Sequence_For_Data_Element;
end if;

-- Advance pointer to next fact in manager. O(1).
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

-- Get a fact. O(1) time.

A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item

```

```

-- MODULE NUMBER: TBD
-- DESCRIPTION:
--      When the flag Type_Of_Facts_Flag is set to true, it means the
--      client procedure wants all the facts that are necessary for
--      the .esm file. If the flag is false, then the facts for
--      the expert system are returned. Facts of the same type have
--      the same format no matter where they are destined. In this
--      case, the historical name is but a single fact. Future
--      modifications to SATool II could include more information in
--      the Historical_Activity_Manager however, thus this procedure
--      is of use.
-- historical tuple facts: (retrieved when creating a .esm file or
--                          when performing check syntax)
--      1) a predefined attribute name (historical-name)
--      2) the historical name (if the name is null, the word 'null' is
--         placed in the field.
-- ALGORITHM: All simple O(1) statements and 2 O(1) procedure calls.
-- PASSED VARIABLES: Type_Facts_Flag, Fact_Manager
-- RETURNS: None
-- GLOBAL VARIABLES USED: None
-- GLOBAL VARIABLES CHANGED: None
-- FILES READ: None
-- FILES WRITTEN: None
-- HARDWARE INPUT: None
-- HARDWARE OUTPUT: None
-- MODULES CALLED: None
-- CALLING MODULES: TBD
-- ORDER-OF: O(1)
-- AUTHOR(S): Min-fuh Shyong
-- HISTORY: None (Initial Implementation)
-----

```

```

procedure Retrieve_Historical_Activity_Facts
  (Type_Facts_Flag : in boolean;
   Fact_Manager    : in out
   Environment_Types.Fact_Buffer_Package.Manager_Type) is

  -- Local Declaration --

  Fact_Pointer : Environment_Types.Fact_Buffer_Package.Iterator_Type;
  A_Fact       : Environment_Types.Fact_String_Type;
  Blank_Fact   : Environment_Types.Fact_String_Type := (others => ' ');

```

```

  -- Historical Activity Declarations --

  Historical_Activity_Record :
    Historical_Activity_Class.Historical_Activity_Record_Type;
  Historical_Activity_Pointer:
    Historical_Activity_Manager.Historical_Activity_Pointer_Type;

```

```

begin

    -- Clear the passed in fact_manager --

Environment_Types.Fact_Buffer_Package.Clear(Fact_Manager);

    -- Reset --

Historical_Activity_Manager.Reset_Historical_Activity_Iterator;

    -- take facts --

while not Historical_Activity_Manager.Historical_Activity_Iterator_Done loop

    Historical_Activity_Record :=
        Historical_Activity_Manager.Value_Of_Historical_Activity_At_Iterator;

    A_Fact          := Blank_Fact;
    A_Fact(1..16)   := "historical-tuple";
    A_Fact(17)      := ' ';
    A_Fact(18..42)  := Historical_Activity_Record.Project;
    A_Fact(43)      := ' ';
    A_Fact(44..63) := Historical_Activity_Record.Activity_Number;

    -- Store the facts --

Environment_Types.Fact_Buffer_Package.Add_Item
    (A_Fact, Fact_Manager, Fact_Pointer);

    Historical_Activity_Manager.Advance_Iterator_To_Next_Historical_Activity;

end loop;
end Retrieve_Historical_Activity_Facts;

```

```

-----
-- DATE: 12/06/90                                     --
-- VERSION: 1.0                                         --
-- NAME:          ***PROCEDURE RESTORE HISTORICAL AVTIVITY FACTS***      --
-- MODULE NUMBER: TBD                                   --
-- DESCRIPTION: Restores the Historical Activity facts into the          --
-- Historical Activity Manager                                           --
-- ALGORITHM: A single while loop controls the execution with an        --
-- embedded call to an O(i) procedure.                                  --
-- PASSED VARIABLES: The_Fact_Buffer (contains the Historical            --
--                      Activity facts)                                  --

```

```

-- RETURNS: None
-- GLOBAL VARIABLES USED: None
-- GLOBAL VARIABLES CHANGED: None
-- FILES READ: None
-- FILES WRITTEN: None
-- HARDWARE INPUT: None
-- HARDWARE OUTPUT: None
-- MODULES CALLED: None
-- CALLING MODULES: TBD
-- ORDER-OF:  $O(i * i)$  where  $i$  is the number of facts in the fact
-- buffer which should be the same as the no. of Historical Activity
-- facts.
-- Note that all string slice operations are modeled as  $O(1)$  time.
-- AUTHOR(S): Min-fuh Shyong
-- HISTORY: None (Initial Implementation)
-----

```

```

procedure Restore_Historical_Activity_Facts
  (The_Fact_Buffer: in
  Environment_Types.Fact_Buffer_Package.Manager_Type) is

```

```

  -- Local Declaration --

```

```

Fact_Pointer   : Environment_Types.Fact_Buffer_Package.Iterator_Type;
A_Fact         : Environment_Types.Fact_String_Type;
First_Char     : natural := 0;
Char_Position  : natural := 0;
Temp_Pos       : natural := 0;

```

```

  -- Historical Activity Declarations --

```

```

Historical_Activity_Record :
  Historical_Activity_Class.Historical_Activity_Record_Type;
Historical_Activity_Pointer:
  Historical_Activity_Manager.Historical_Activity_Pointer_Type;

```

```

  -- add new variable --

```

```

Null_Historical_Activity_Record      :
  Historical_Activity_Class.Historical_Activity_Record_Type;

```

```

begin

```

```

  -- check for empty buffer of facts. If empty, do nothing. --

```

```

if Environment_Types.Fact_Buffer_Package.Is_Empty(The_Fact_Buffer) then
  return;
end if;

```

```

-- Initialize iterator to first historical activity facts --
Environment_Types.Fact_Buffer_Package.Initialize_Iterator
(Fact_Pointer, The_Fact_Buffer);

-- Engage loop to extract the facts from a buffer --
-- This loop is O(i) time.
-- Where i is the number of facts in the buffer

while not Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) loop
-- Get a record
A_Fact := Environment_Types.Fact_Buffer_Package.Value_Of_Item
(Fact_Pointer) ;

-- Since we put the information in the string, we know the
-- exact columns where information should be.
-- All String assignments are modeled as O(1);

-- Insure the fields are all blanks

Historical_Activity_Record := Null_Historical_Activity_Record;

-- Retrieve the facts

Historical_Activity_Record.Project := A_Fact(18..42);
Historical_Activity_Record.Activity_Number := A_Fact(44..63);

-- load this fact back into Historical Activity Manager

Historical_Activity_Manager.Create_Historical_Activity
(Historical_Activity_Record, Historical_Activity_Pointer);

-- Advance pointer --
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

end loop;
end Restore_Historical_Activity_Facts;

```

```

-----
-- DATE: 12/01/90 --
-- VERSION: 1.0 --
-- NAME: *** RETRIEVE CALLS RELATION FACTS *** --
-- MODULE NUMBER: TBD --
-- DESCRIPTION: --
-- When the flag Type_Of_Facts_Flag is set to true, it means the --
-- client procedure wants all the facts that are necessary for --
-- the .esm file. If the flag is false, then the facts for --
-- the expert system are returned. Facts of the same type have --
-- the same format no matter where they are destined. In this --

```



```

--      case, the calls name is but a single fact.
-- calls relation tuple facts: (calls-relation-tuple  Activity
--                               History-tuple)
--      where history is another tuple in Historical_Activity
-- ALGORITHM: All simple O(1) statements and 2 O(1) procedure calls.
-- PASSED VARIABLES: Type_Facts_Flag, Fact_Manager
-- RETURNS: None
-- GLOBAL VARIABLES USED:  None
-- GLOBAL VARIABLES CHANGED: None
-- FILES READ : None
-- FILES WRITTEN: None
-- HARDWARE INPUT: None
-- HARDWARE OUTPUT: None
-- MODULES CALLED : None
-- CALLING MODULES: TBD
-- ORDER-OF:  O(1)
-- AUTHOR(S): Min-fuh Shyong
-- HISTORY: None (Initial Implementation)
-----

```

```

procedure Retrieve_Calls_Relation_Facts
  (Type_Facts_Flag : in boolean;
   Fact_Manager    : in out
   Environment_Types.Fact_Buffer_Package.Manager_Type) is

  -- local declaration --

Fact_Pointer : Environment_Types.Fact_Buffer_Package.Iterator_Type;
A_Fact       : Environment_Types.Fact_String_Type;
Blank_Fact   : Environment_Types.Fact_String_Type := (others => ' ');

  -- calls related declarations --

Calls_Relation_Record : Calls_Relation_Class.Calls_Relation_Record_Type;
Calls_Relation_Pointer : Calls_Relation_Manager.Calls_Relation_Pointer_Type;

begin

  -- clear the buffer --

Environment_Types.Fact_Buffer_Package.Clear(Fact_Manager);

  -- reset --

Calls_Relation_Manager.Reset_Calls_Relation_Tuple_Iterator;

  -- teke facts --

while not  Calls_Relation_Manager.Calls_Relation_Tuple_Iterator_done loop

  Calls_Relation_Record :=

```

```

Calls_Relation_Manager.Value_Of_Calls_Relation_Tuple_At_Iterator;

A_Fact      := Blank_Fact;
A_Fact(1..20) := "calls-relation-tuple";
A_Fact(21)   := ' ';
A_Fact(22..46) := Calls_Relation_Record.Activity;
A_Fact(47)   := ' ';
A_Fact(48..72) := Calls_Relation_Record.History_Tuple.Project;
               -- with one more . extension to get the
               -- nested record History_Tuple.Project
A_Fact(73)   := ' ';
A_Fact(74..93) := Calls_Relation_Record.History_Tuple.Activity_Number;

Environment_Types.Fact_Buffer_Package.Add_Item
  (A_Fact, Fact_Manager, Fact_Pointer);

Calls_Relation_Manager.Advance_Iterator_To_Next_Calls_Relation_Tuple;

end loop;

end Retrieve_Calls_Relation_Facts;

-----
-- DATE: 12/06/90
-- VERSION: 1.0
-- NAME:      ***PROCEDURE RESTORE CALLS RELATION FACTS***
-- MODULE NUMBER: TBD
-- DESCRIPTION: This procedure accepts a buffer of calls relation
-- facts.
-- Restores that information into the calls relation manager.
-- Of special note is that the procedure assumes the facts are in the
-- same order in which they were stored.
-- ALGORITHM: A single while loop controls the execution with an
-- embedded call to an O(i) procedure.
-- PASSED VARIABLES: The_Fact_Buffer (contains the facts)
-- RETURNS: None
-- GLOBAL VARIABLES USED: None
-- GLOBAL VARIABLES CHANGED: None
-- FILES READ: None
-- FILES WRITTEN: None
-- HARDWARE INPUT: None
-- HARDWARE OUTPUT: None
-- MODULES CALLED: None
-- CALLING MODULES: Essential_IO.Restore_Project
-- ORDER-OF: order of is O(a * max (x, a)) where a is the
-- number of calls, x is the number of lines in a historical activity
-- Note that all string slice operations are modeled as O(1) time.
-- AUTHOR(S): Min-fuh Shyong
-- HISTORY: None (Initial Implementation)
-----

```

```

procedure Restore_Calls_Relation_Facts
  (The_Fact_Buffer : in
   Environment_Types.Fact_Buffer_Package.Manager_Type) is

  -- Local Declarations --

  Fact_Pointer: Environment_Types.Fact_Buffer_Package.Iterator_Type;
  A_Fact: Environment_Types.Fact_String_Type;

  First_Char: natural:= 0;
  Char_Position: natural:= 0;
  Temp_Pos: natural:= 0;
  More_Descriptions_Flag: boolean;

  -- Calls Relation Related Declarations --

  Calls_Relation_Record: Calls_Relation_Class.Calls_Relation_Record_Type;
  Calls_Relation_Pointer: Calls_Relation_Manager.Calls_Relation_Pointer_Type;

  Null_Calls_Relation_Record: Calls_Relation_Class.Calls_Relation_Record_Type;

  --The_Iterator: Environment_Types.Text_Buffer_Package.Iterator_Type;
  --A_Description_Line: Environment_Types.DD_Text_Type;
  --A_Child: Environment_Types.DD_Field_Type;
  --Found_Flag: boolean:= False;
  --Result_Flag: boolean;

  -- Exception --
  -- This exception is declared here because the Essential IO package does
  -- not check to see the facts are in any specific order.
  --Invalid_Fact_Sequence_For_Calls_Relation: exception;
  --Activity_Hierarchy_Error_During_Restore: exception;

begin
  -- Check for empty buffer of facts. If empty, do nothing.
  if Environment_Types.Fact_Buffer_Package.Is_Empty(The_Fact_Buffer) then
    return;
  end if;

  -- Initialize iterator to first tuple fact

  Environment_Types.Fact_Buffer_Package.Initialize_Iterator
    (Fact_Pointer, The_Fact_Buffer);

  -- Engage loop to extract the cassl relation facts from a buffer
  -- one at a time
  while not Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) loop
    -- Get a record.

```

```

A_Fact:= Environment_Types.Fact_Buffer_Package.Value_Of_Item
      (Fact_Pointer);

Calls_Relation_Record := Null_Calls_Relation_Record;

Calls_Relation_Record.Activity := A_Fact(22..46);
Calls_Relation_Record.History_Tuple.Project := A_Fact(48..72);
Calls_Relation_Record.History_Tuple.Activity_Number := A_Fact(74..93);

Calls_Relation_Manager.Create_Calls_Relation_Tuple
      (Calls_Relation_Record, Calls_Relation_Pointer);

      Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);
end loop;
end Restore_Calls_Relation_Facts;

-----
-- DATE: 12/03/90 --
-- VERSION: 1.0 --
-- NAME: *** RETRIEVE CONSISTS OF RELATION *** --
-- MODULE NUMBER: TBD --
-- DESCRIPTION: --
--      When the flag Type_Of_Facts_Flag is set to true, it means the --
--      client procedure wants all the facts that are necessary for --
--      the .esm file. If the flag is false, then the facts for --
--      the expert system are returned. Facts of the same type have --
--      the same format no matter where they are destined. In this --
--      case, the historical name is but a single fact. Future --
--      modifications to SATool II could include more information in --
--      the Historical_Activity_Manager however, thus this procedure --
--      is of use. --
-- consists of relation facts: (retrieved when creating a .esm file --
-- or when performing check syntax --
--      1) a predefined attribute name (consists-of-name) --
--      2) the consists of name (if the name is null, the word 'null' is --
--      placed in the field. --
-- ALGORITHM: All simple O(1) statements and 2 O(1) procedure calls. --
-- PASSED VARIABLES: Type_Facts_Flag, Fact_Manager --
-- RETURNS: None --
-- GLOBAL VARIABLES USED: None --
-- GLOBAL VARIABLES CHANGED: None --
-- FILES READ: None --
-- FILES WRITTEN: None --
-- HARDWARE INPUT: None --
-- HARDWARE OUTPUT: None --
-- MODULES CALLED: None --
-- CALLING MODULES: TBD --
-- ORDER-OF: O(1) --

```

```
-- AUTHOR(S): Min-fuh Shyong --
-- HISTORY: None (Initial Implementation) --
-----
```

```
procedure Retrieve_Consists_Of_Relation_Facts
  (Type_Facts_Flag : in boolean;
   Fact_Manager    : in out
    Environment_Types.Fact_Buffer_Package.Manager_Type) is

  -- local declaration --

  Fact_Pointer : Environment_Types.Fact_Buffer_Package.Iterator_Type;
  A_Fact       : Environment_Types.Fact_String_Type;
  Blank_Fact   : Environment_Types.Fact_String_Type := (others => ' ');

  -- consists of declarations --

  Consists_Of_Relation_Record :
    Consists_Of_Relation_Class.Consists_Of_Relation_Record_Type;

  Consists_Of_Relation_Pointer :
    Consists_Of_Relation_Manager.Consists_Of_Relation_Pointer_Type;

begin

  -- Clear the passed in fact_manager --

  Environment_Types.Fact_Buffer_Package.Clear(Fact_Manager);

  -- Reset --

  Consists_Of_Relation_Manager.Reset_Consists_Of_Relation_Tuple_Iterator;

  -- take facts --

  while not Consists_Of_Relation_Manager.
    Consists_Of_Relation_Tuple_Iterator_Done loop

    Consists_Of_Relation_Record:=
      Consists_Of_Relation_Manager.
        Value_Of_Consists_Of_Relation_Tuple_At_Iterator;

    A_Fact      := Blank_Fact;
    A_Fact(1..16) := "consists-of-name";
    A_Fact(17)  := ' ';
    A_Fact(18..23) :=
      Padded_String(integer'image(Consists_Of_Relation_Record.Consists_Of_Id), 6);
    A_Fact(24) := ' ';
    A_Fact(25..49) := Consists_Of_Relation_Record.Parent;
    A_Fact(50) := ' ';
```

```

A_Fact(51..75) := Consists_Of_Relation_Record.Child;

Environment_Types.Fact_Buffer_Package.Add_Item
  (A_Fact, Fact_Manager, Fact_Pointer);

Consists_Of_Relation_Manager.
  Advance_Iterator_To_Next_Consists_Of_Relation_Tuple;

end loop;

end Retrieve_Consists_Of_Relation_Facts;

-----
-- DATE: 12/06/90 --
-- VERSION: 1.0 --
-- NAME: ***PROCEDURE RESTORE CONSISTS OF RELATION FACTS*** --
-- MODULE NUMBER: TBD --
-- DESCRIPTION: Restores the consists of relation facts into the --
-- Consists Of Relation Manager --
-- ALGORITHM: A single while loop controls the execution with an --
-- embedded call to an O(i) procedure. --
-- PASSED VARIABLES: The_Fact_Buffer (contains the consists of --
-- relation facts) --
-- RETURNS: None --
-- GLOBAL VARIABLES USED: None --
-- GLOBAL VARIABLES CHANGED: None --
-- FILES READ: None --
-- FILES WRITTEN: None --
-- HARDWARE INPUT: None --
-- HARDWARE OUTPUT: None --
-- MODULES CALLED: None --
-- CALLING MODULES: TBD --
-- ORDER-OF: O(i * i) where i is the number of facts in the fact --
-- buffer which should be the same as the no. of Consists of relation --
-- facts. --
-- Note that all string slice operations are modeled as O(1) time. --
-- AUTHOR(S): Min-fuh Shyong --
-- HISTORY: None (Initial Implementation) --
-----

procedure Restore_Consists_Of_Relation_Facts
  (The_Fact_Buffer: in
    Environment_Types.Fact_Buffer_Package.Manager_Type) is

  -- Local Declaration --

  Fact_Pointer : Environment_Types.Fact_Buffer_Package.Iterator_Type;
  A_Fact : Environment_Types.Fact_String_Type;
  First_Char : natural := 0;

```

```

Char_Position : natural := 0;
Temp_Pos      : natural := 0;

-- Consists Of Relation Declarations --

Consists_Of_Relation_Record :
  Consists_Of_Relation_Class.Consists_Of_Relation_Record_Type;
Consists_Of_Relation_Pointer:
  Consists_Of_Relation_Manager.Consists_Of_Relation_Pointer_Type;

-- add new variable --
Null_Consists_Of_Relation_Record :
  Consists_Of_Relation_Class.Consists_Of_Relation_Record_Type;

begin
  -- check for empty buffer of facts. If empty, do nothing. --

if Environment_Types.Fact_Buffer_Package.Is_Empty(The_Fact_Buffer) then
  return;
end if;

  -- Initialize iterator to first consists of relation facts --

Environment_Types.Fact_Buffer_Package.Initialize_Iterator
  (Fact_Pointer, The_Fact_Buffer);

  -- Engage loop to extract the facts from a buffer --
  -- This loop is O(i) time.
  -- Where i is the number of facts in the buffer

while not Environment_Types.Fact_Buffer_Package.Is_Done(Fact_Pointer) loop
  -- Get a record
  A_Fact := Environment_Types.Fact_Buffer_Package.Value_Of_Item
    (Fact_Pointer);

  -- Since we put the information in the string, we know the
  -- exact columns where information should be.
  -- All String assignments are modeled as O(1);

  -- Insure the fields are all blanks

  Consists_Of_Relation_Record := Null_Consists_Of_Relation_Record;

  -- Retrieve the facts

  Consists_Of_Relation_Record.Consists_Of_Id :=
    integer'value(A_Fact(18..23));
  Consists_Of_Relation_Record.Parent := A_Fact(25..49);

```

```

Consists_Of_Relation_Record.Child := A_Fact(51..75);

-- load this fact back into Consists Of Relation Manager

Consists_Of_Relation_Manager.Create_Consists_Of_Relation_Tuple
  (Consists_Of_Relation_Record, Consists_Of_Relation_Pointer);

-- Advance pointer --
Environment_Types.Fact_Buffer_Package.Get_Next(Fact_Pointer);

end loop;
end Restore_Consists_Of_Relation_Facts;

end Essential_Fact_Uilities;

```


Appendix C. CLIPS RULE BASE

```
*****
;;           Essential Subsystem Rule Base           ;;
;;           =====                               ;;
;; File Name: satool2.clp                           ;;
;; Date Last Updated: 24 May 1991                    ;;
;; Author: Min-fuh Shyong, GCS-91j                   ;;
;; Points of Contact: Dr. Gary Lamont                 ;;
;; DESCRIPTION:                                       ;;
;;   This file contains the rule base used by        ;;
;; the CLIPS/Ada expert system portion of the Essential ;;
;; Subsystem. The idea was initiated by Terry Kitchen in ;;
;; his thesis but needs to be expanded and completed for the ;;
;; follow on researchers. This subsystem is to eventually be ;;
;; integrated with another system to form SATool II, which ;;
;; with another system to form SATool II, which is an Ada ;;
;; is an Ada based IDEFO development tool.            ;;
;; PURPOSE:                                           ;;
;;   The purpose of this rule base is to check the   ;;
;; syntactic features of an IDEFO model whose representation ;;
;; has been converted to CLIPS readable facts.        ;;
;; METHODOLOGY:                                       ;;
;;   Whenever the "check syntax" option is chosen within ;;
;; the Essential Subsystem main menu, this rule base is loaded;;
;; into the working memory of the CLIPS/Ada expert system. ;;
;; The same option also begins the "recognize-act" cycle of ;;
;; the CLIPS inference engine which uses the rules below to ;;
;; "match" the LHS of rules with facts, resolve conflicts ;;
;; among eligible rules, and then fire the RHS of rules, until;;
;; no rules are eligible to fire. This file must be within ;;
;; SCOPE:                                             ;;
;;   At the present time, this rule base checks the   ;;
;; syntactical features associated with the "essential" data ;;
;; of an IDEFO model.                                ;;
;; RULES AND THEIR FUNCTION:                         ;;
;;   The following IDEFO syntax checking rules are completed: ;;
;;   ;;
;; 1. Each activity is checked to ensure it has at least one ;;
;;    output and one control.                            ;;
;; 2. For each activity, the number of its input, output, ;;
;;    control and mechanisms is checked to suggest that they ;;
;;    are not more than 5.                                ;;
;; 3. The project is checked to ensure a name is given for ;;
;;    the project.                                         ;;
;; 4. Each activity is checked to ensure an activity number ;;
;;    is assigned. No duplicated activity name is also ;;
;;    checked.                                             ;;
;; 5. Each activity is checked to ensure some description ;;
```

```

;; are associated with that activity. ;;
;; 6. Each data element is checked to make sure that the ;;
;; data name, description are provided. And no duplicated ;;
;; data element name exists. ;;
;; 7. Each parent activity with a child name with it, the ;;
;; child's name must be found. ;;
;; 8. Hierarchical rules for creating boundary arrow facts ;;
;; between any parent with 2, 3, 4, 5, or 6 child ;;
;; activities are implemented. ;;
;; 9. Rules for checking the consistency between those parent ;;
;; diagram and their child diagrams are provided. ;;
;;10. Rule for checking inconsistent icom code between parent ;;
;; and child diagrams. ;;
;;11. Rule to check if any parent activity has more than 6 ;;
;; child diagrams. ;;
;;12. Utility rules builds up the syntax checking ;;
;; environment. ;;
;;13. Boundary icom number consistency checking rules. ;;
;;14. Auxiliary rules supporting the hierarchy checking rules.;;
;; OUTPUT: ;;
;; IDEFO syntax violations cause the user to receive ;;
;; five kinds of messages: ;;
;; 1. CONGRATULATORY: No syntax errors was found. If no ;;
;; syntax error facts was asserted after the ;;
;; rules checking is done, then this message ;;
;; will be presented at the end of all the ;;
;; other messages. ;;
;; 2. ERRORS: Syntax error encountered, syntax error ;;
;; fact will be asserted, program will be ;;
;; halted after all the checkings are done. ;;
;; 3. WARNING: Some features of the users project work ;;
;; were discovered that might cause problem. ;;
;; 4. NOTICE: Reminder to the user that something should ;;
;; be carefully done. ;;
;; 5. SUGGESTION: Suggest the user that further manually ;;
;; recheck might be helpful to find ;;
;; logical errors that cannot be found by ;;
;; the syntax checking rules. ;;
;; *****

```

```

***** Environment Utility rule *****
; These rules does not do the syntax checking functions, but are necessary
; for the syntax checking package.
; -----

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule prints out the necessary headings for the syntax checking
; functions. It is guaranteed by the salience declaration to be fired first.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```



```
; If an activity has no output, no control, than it is an syntax error.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(defrule zero-outputs
  (icom-activity-outputs ?act 0)
  =>
  (printout t "ERROR: Activity " ?act " needs at least 1 output."
    crlf )
  (assert (syntax-error-occurred))
)
```

```
(defrule zero-controls
  (icom-activity-controls ?act 0)
  =>
  (printout t "ERROR: Activity " ?act " needs at least 1 control."
    crlf )
  (assert (syntax-error-occurred))
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Checks if the inputs, mechanisms, controls or outputs of an activity
; is more that 5, than a warning message will be presented.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(defrule too-many-mechs
  (icom-activity-mechanisms ?act-mech ?num-mech)
  (test (> ?num-mech 5))
  =>
  (printout t "WARNING: Activity " ?act-mech " has too many
    mechanisms." crlf)
)
```

```
(defrule too-many-outputs
  (icom-activity-outputs ?act-out ?num-out)
  (act-numb ?act-out ?out-num)
  (test (> ?num-out 5))
  =>
  (printout t "WARNING: Activity " ?out-num " " ?act-out " has too many
    outputs." crlf)
)
```

```
(defrule too-many-controls
  (icom-activity-controls ?act-cont ?num-cont)
  (act-numb ?act-cont ?cont-num)
```



```

(defrule null-activity-description
  (act-desc ?activity null)
  (act-numb ?activity ?num)
  =>
  (printout t "WARNING: Activity number " ?num " " ?activity " needs a
  description." crlf))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; If any activity has a activity number with the last digit more than
; 6, than that means its parent has more than 6 child activities.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule too-many-children-level1
  (act ?act ?end-num)
  (test (> ?end-num 6))
  =>
  (printout t "Waring: activity A0 has more than 6 child diagrams." crlf)
  (printout t "Notice: Please manually check to make sure that there is no" crlf)
  (printout t " such an warning lower that 4 levels of hierarchy." crlf)
  )

(defrule too-many-children-level2
  (act ?act ?num ?end-num)
  (test (> ?end-num 6))
  =>
  (printout t "Waring: activity A"?num " has more than 6 child diagrams." crlf)
  (printout t "Notice: Please manually check to make sure that there is no" crlf)
  (printout t " such an warning lower that 4 levels of hierarchy." crlf)
  )

(defrule too-many-children-level3
  (act ?act ?n1 ?n2 ?end-num)
  (test (> ?end-num 6))
  =>
  (printout t "Waring: activity A"?n1 ?n2 " has more than 6 child diagrams." crlf)
  (printout t "Notice: Please manually check to make sure that there is no" crlf)
  (printout t " such an warning lower that 4 levels of hierarchy." crlf)
  )

; ***** rules for hierarchical boundary consistency checks *****
;
; -----
; The boundary data element name and icom code of a parent activity
; must be consistent with its child diagrams. Those rules create a
; set of boundary facts to be checked for their consistency.
; The assumption made here is that any parent activity should not have
; more than six child diagrams. So the rules are implemented
; to create boundary facts for parent activity with 2, 3, 4, 5, and 6
; child diagrams separately.
; Another set of rules will be used to check the created boundary facts
; for activities with different or same number of child activities.

```

;-----

```
;;;;;;;;;;
; These rules create boundary facts for a parent activity with
; two child diagrams. The first level rule creates those initial
; boundary facts, the second level rules clear the data in child
; diagrams that are not boundary ; data in contrast with their
; brother diagrams.
;;;;;;;;;;
```

```
(defrule parent-2child
  (declare (salience 100))
  ?f1<-(act-has-child ?parent2 ?child1&~null)
  ?f2<-(act-has-child ?parent2 ?child2&~?child1&~null)
  (not (act-has-child ?parent2
    ?child3&~?child2&~?child1&~null) )
  =>
  (retract ?f1 ?f2)
  (assert (parent2 ?parent2 ?child1 ?child2))
)
```

```
(defrule parent2-boundary
  (parent2 ?parent2 ?child1 ?child2)
  (icom-tuple ?parent2 ?p-data ?p-rel ?)
  =>
  (assert (parent2-boundary ?parent2 ?p-data ?p-rel))
)
```

```
(defrule child2-boundary-child1
  (parent2 ?parent2 ?child1 ?child2)
  (icom-tuple ?child1 ?c1-data ?c1-rel ?)
  =>
  (assert (child2-boundary ?parent2 ?child1 ?c1-data ?c1-rel))
)
```

```
(defrule child2-boundary-child2
  (parent2 ?parent2 ?child1 ?child2)
  (icom-tuple ?child2 ?c2-data ?c2-rel ?)
  =>
  (assert (child2-boundary ?parent2 ?child2 ?c2-data ?c2-rel))
)
```

```
;;;;;;;;;;
; The (child2-boundary ..... ) facts created by the previous rule
; are only initial boundary facts, which means they still
; have all the data element in the facts. But the data shared by any two
; different child activities with different icom code will not be boundary
```

```

; arrows for the child activities.
;   They should be retracted from the
; facts already created before the boundary checking actually performs.
; And they must be executed after all the initial boundary facts are
; already created. This is guaranteed by a higher salience declaration in
; the previous rule.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule clear-2child-mid
  ?f1<-(child2-boundary ?parent2 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child2-boundary ?parent2 ?child2~?child1 ?c1-data ?c2-rel~?c1-rel)
  =>
  (retract ?f1)
  (retract ?f2)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule erase one of the duplicated boundary arrow
; for the icom number check.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule remove-2child-2boundary
  ?f1<- (child2-boundary ?parent2 ?child1 ?c1-data ?c1-rel)
        (child2-boundary ?parent2 ?child2~?child1 ?c1-data ?c1-rel)
  =>
  (retract ?f1)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; If an intermediate data consists subcomponents, it should be
; retracted as well.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule rid-2child-2consists
  ?f1<-(child2-boundary ?parent2 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child2-boundary ?parent2 ?child1 ?c2-data~?c1-data ?c2-rel~?c1-rel)
  ?f3<-(child2-boundary ?parent2 ?child-p~?child1 ?cp-data~?c1-data~?c2-data
                    ?cp-rel~?c1-rel~?c2-rel)
        (consists-of-name ? ?cp-data ?c2-data)
        (consists-of-name ? ?cp-data ?c1-data)
  =>
  (retract ?f1 ?f2 ?f3)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; For those parent activity with two child diagrams,
; if a parent boundary data can't be found in the child boundary data
; or the parent data is not a parent-data of the child data,
; than parent inconsistency occurred.

```



```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule check-2child-parent
  (declare (salience -5))
  (parent2-boundary ?p-name ?p-data ?p-rel)
  (not (consists-of-name ? ?p-data ?c-data))
  (not (child2-boundary ?p-name ?childior2 ?p-data ?c-rel))
=>
  (printout t "ERROR: Data inconsistency between parent activity" crlf)
  (printout t "    " ?p-name " data " ?p-rel " " ?p-data " and its" crlf)
  (printout t "        child diagrams." crlf)
  (assert (syntax-error-occurred))
)

```

```

(defrule check-2child-parent-consists
  (declare (salience -6))
  (parent2-boundary ?p-name ?p-data ?p-rel)
  (consists-of-name ? ?p-data ?c-data)
  (not (child2-boundary ?p-name ?child2 ?c-data ?c2-rel))
=>
  (printout t "ERROR: Data inconsistency between parent activity
    "?p-name " data " ?p-rel " " ?p-data " and its child
    diagrams." crlf)
  (assert (syntax-error-occurred))
)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; If a parent finds a child with same data but different
; relation, then it is an icom inconsistency.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defrule check-2child-parent-icom
  (declare (salience -5))
  (parent2-boundary ?p-name ?p-data ?p-rel)
  (child2-boundary ?p-name ?child1 ?p-data ?c-rel)
  (test (neq ?p-rel ?c-rel))
=>
  (printout t "ERROR: icom inconsistency between activity " crlf)
  (printout t "    " ?p-name " and its child diagram " ?child1 "." crlf)
  (assert (syntax-error-occurred))
)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule checks if a data in child is not in their parent
; than child inconsistency with its parent occurred.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defrule check-2child-child
  (declare (salience -5))
  (child2-boundary ?p-name ?child1 ?c1-data ?c1-rel)
  (not (consists-of-name ? ?p-data ?c-data))
  (not (parent2-boundary ?p-name ?c1-data ?p-rel))
=>
  (printout t "ERROR: Data inconsistency between child
  activity " ?child1 " data '" ?c1-rel "' " ?c1-data
  " and its parent." crlf)
  (assert (syntax-error-occurred))
)

;*****
;                               Boundary icom number rules
;                               =====
; Those hierarchical rules will create a set of facts calculating
; and accumulating the control, output, input, and mechanisms numbers
; for each parent activity and their child activities boundary icom facts.
;
; Note that these rules are dependants of those used to create boundary
; facts for each pair of parent and child activities. So their salience must
; be lower to be fired later after those boundary facts have already
; been created.
; *****

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;                               Parent with 2 child diagrams
;                               -----
; The initial icom number will be build up by these rules;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
(defrule parent2-icom-c
  (declare (salience -2))
  (parent2-boundary ?p2-name ?c1 ?c2 ?p2-data c)
=>
  (assert (parent2-icom ?p2-name ?p2-data control 1))
)

(defrule parent2-icom-o
  (declare (salience -2))
  (parent2-boundary ?p2-name ?c1 ?c2 ?p2-data o)
=>
  (assert (parent2-icom ?p2-name ?p2-data output 1))
)

(defrule parent2-icom-i
  (declare (salience -2))
  (parent2-boundary ?p2-name ?c1 ?c2 ?p2-data i)
=>
  (assert (parent2-icom ?p2-name ?p2-data input 1))
)

```

```

(defrule parent2-icom-m
  (declare (salience -2))
  (parent2-boundary ?p2-name ?c1 ?c2 ?p2-data m)
  =>
  (assert (parent2-icom ?p2-name ?p2-data mech 1))
  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; As the icom facts are created, these rules will add up
; the total number of icom for each activity.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule parent2-control-add
  (declare (salience -3))
  ?f1<-(parent2-icom ?p2-name ?data1 control ?one)
  ?f2<-(parent2-icom ?p2-name ?data2 control ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent2-icom ?p2-name =(gensym) control ?total))
  )

; -----

(defrule parent2-output-add
  (declare (salience -3))
  ?f1<-(parent2-icom ?p2-name ?data1 output ?one)
  ?f2<-(parent2-icom ?p2-name ?data2 output ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent2-icom ?p2-name =(gensym) output ?total))
  )

; -----

(defrule parent2-input-add
  (declare (salience -3))
  ?f1<-(parent2-icom ?p2-name ?data1 input ?one)
  ?f2<-(parent2-icom ?p2-name ?data2 input ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent2-icom ?p2-name =(gensym) input ?total))
  )

```

```

; -----
(defrule parent2-mech-add
  (declare (salience -3))
  ?f1<-(parent2-icom ?p2-name ?data1 mech ?one)
  ?f2<-(parent2-icom ?p2-name ?data2 mech ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent2-icom ?p2-name =(gensym) mech ?total))
  )

; ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

(defrule child2-icom-c
  (declare (salience -2))
  (child2-boundary ?c2-parent ?c2-name ?c2-data c)
  =>
  (assert (child2-icom ?c2-parent ?c2-data control 1))
  )

(defrule child2-icom-o
  (declare (salience -2))
  (child2-boundary ?c2-parent ?c2-name ?c2-data o)
  =>
  (assert (child2-icom ?c2-parent ?c2-data output 1) )
  )

(defrule child2-icom-i
  (declare (salience -2))
  (child2-boundary ?c2-parent ?c2-name ?c2-data i)
  =>
  (assert (child2-icom ?c2-parent ?c2-data input 1) )
  )

(defrule child2-icom-m
  (declare (salience -2))
  (child2-boundary ?c2-parent ?c2-name ?c2-data m)
  =>
  (assert (child2-icom ?c2-parent ?c2-data mech 1) )
  )

; -----

(defrule child2-control-add
  (declare (salience -3))
  ?f1<-(child2-icom ?c2-parent ?data1 control ?one)
  ?f2<-(child2-icom ?c2-parent ?data2 control ?n)
  (test (neq ?data1 ?data2))

```

```

=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child2-icom ?c2-parent =(gensym) control ?total))
)

; -----

(defrule child2-output-add
(declare (salience -3))
?f1<-(child2-icom ?c2-parent ?data1 output ?one)
?f2<-(child2-icom ?c2-parent ?data2 output ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child2-icom ?c2-parent =(gensym) output ?total))
)

; -----

(defrule child2-input-add
(declare (salience -3))
?f1<-(child2-icom ?c2-parent ?data1 input ?one)
?f2<-(child2-icom ?c2-parent ?data2 input ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child2-icom ?c2-parent =(gensym) input ?total))
)

; -----

(defrule child2-mech-add
(declare (salience -3))
?f1<-(child2-icom ?c2-parent ?data1 mech ?one)
?f2<-(child2-icom ?c2-parent ?data2 mech ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child2-icom ?c2-parent =(gensym) mech ?total))
)

;*****
;          Check Parent with 2 child  boundary icom number consistency.
; If the number of boundary icom for a parent activity is not the same
; with its child activities. A warning will be raised.
;-----

```

```

(defrule check-parent-2child-control
  (declare (salience -6))
  ?f1<-(parent2-icom ?p2-name ? control ?p)
  ?f2<-(child2-icom ?p2-name ? control ?c)
  (test (!= ?p ?c))
  =>
  (retract ?f1 ?f2)
  (if (> ?p ?c)
    then
      (bind ?pd (- ?p ?c))
      (printout t "WARNING, there might be an ERROR: The number of boundary controls" crlf)
      (printout t "  of parent activity "?p2-name" is "?pd" control(s) more than " crlf)
      (printout t "  its child activities." crlf)
      (printout t "  Are there 'consists of' data items at boundary?" crlf)
      (printout t "  Please recheck the syntax." crlf)
    else
      (bind ?cd (- ?c ?p))
      (printout t "WARNING, there might be an ERROR: The number of boundary controls" crlf)
      (printout t "  of the parent activity "?p2-name" is "?cd" control(s) less " crlf)
      (printout t "  than its child boundary controls." crlf)
      (printout t "  Are there 'consists of' data items at boundary?" crlf)
      (printout t "  Please recheck the syntax." crlf)
  ) )

(defrule check-parent-2child-output
  (declare (salience -6))
  ?f1<-(parent2-icom ?p2-name ? output ?p)
  ?f2<-(child2-icom ?p2-name ? output ?c)
  (test (!= ?p ?c))
  =>
  (retract ?f1 ?f2)
  (if (> ?p ?c)
    then
      (bind ?pd (- ?p ?c))
      (printout t "WARNING, there might be an ERROR: The number of boundary outputs" crlf)
      (printout t "  of parent activity " ?p2-name " is " ?pd " output(s) more " crlf)
      (printout t "  than its child activities." crlf)
      (printout t "  Are there 'consists of' data items at boundary?" crlf)
      (printout t "  Please recheck the syntax." crlf)
    else
      (bind ?cd (- ?c ?p))
      (printout t "WARNING, there might be an ERROR: The number of boundary outputs" crlf)
      (printout t "  of the parent activity " ?p2-name " is " ?cd " output(s) less " crlf)
      (printout t "  than its child boundary outputs." crlf)
      (printout t "  Are there 'consists of' data items at boundary?" crlf)
      (printout t "  Please recheck the syntax." crlf)
  ) )

```

```

(defrule check-parent-2child-input
  (declare (salience -6))
  ?f1<-(parent2-icom ?p2-name ? input ?p)
  ?f2<-(child2-icom ?p2-name ? input ?c)
  (test (!= ?p ?c))
  =>
  (retract ?f1 ?f2)
  (if (> ?p ?c)
    then
      (bind ?pd (- ?p ?c))
      (printout t "WARNING, there might be an ERROR: The number of boundary inputs" crlf)
      (printout t "  of parent activity " ?p2-name " is " ?pd " input(s) more " crlf)
      (printout t "  than its child activities." crlf)
      (printout t "  Are there 'consists of' data items at boundary?" crlf)
      (printout t "  Please recheck the syntax." crlf)
    else
      (bind ?cd (- ?c ?p))
      (printout t "WARNING, there might be an ERROR: The number of boundary inputs" crlf)
      (printout t "  of the parent activity " ?p2-name " is " ?cd " input(s) less " crlf)
      (printout t "  than its child boundary inputs." crlf)
      (printout t "  Are there 'consists of' data items at boundary?" crlf)
      (printout t "  Please recheck the syntax." crlf)
  ) )

```

```

(defrule check-parent-2child-mech
  (declare (salience -6))
  ?f1<-(parent2-icom ?p2-name ? mech ?p)
  ?f2<-(child2-icom ?p2-name ? mech ?c)
  (test (!= ?p ?c))
  =>
  (retract ?f1 ?f2)
  (if (> ?p ?c)
    then
      (bind ?pd (- ?p ?c))
      (printout t "WARNING, there might be an ERROR: The number of boundary mechanisms" crlf)
      (printout t "  of parent activity " ?p2-name " is " ?pd " mechanism(s) more " crlf)
      (printout t "  than its child activities." crlf)
      (printout t "  Are there 'consists of' data items at boundary?" crlf)
      (printout t "  Please recheck the syntax." crlf)
    else
      (bind ?cd (- ?c ?p))
      (printout t "WARNING, there might be an ERROR: The number of boundary mechanisms" crlf)
      (printout t "  of the parent activity " ?p2-name " is " ?cd " mechanism(s) less " crlf)
      (printout t "  its child child boundary mechanisms." crlf)
      (printout t "  Are there 'consists of' data items at boundary?" crlf)
      (printout t "  Please recheck the syntax." crlf)
  ) )

```



```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; These 2 rules below will erase the duplicated data element
; in the:
; (child3-boundary facts ....)
; The only possible data element we want to erase is
; any data that is shared by 2 or 3 activities but with different
; icom code.
; CONDITION:
;   1. Any two activities are sharing a data element but
;       with different icom relations.
;   2. All three activities are sharing a data element but
;       with different icom relations.
; With the declaration of salience, we may assure that
; any data element shared by all 3 child activities will be
; erased first.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defrule clear-3child-3mid
  ?f1<-(child3-boundary ?parent3 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child3-boundary ?parent3 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
  ?f3<-(child3-boundary ?parent3 ?child3&~?child2&~?child1
        ?c1-data ?c3-rel&~?c2-rel&~?c1-rel)

  =>
  (retract ?f1)
  (retract ?f2)
  (retract ?f3)
)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; If a intermediate arrow is the input of one box but also the
; output and input of another two boxes. It must be removed before
; the arrow between the other boxes been removed.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defrule clear-3child-2mid-1
  (child3-boundary ?paernt3 ?child1 ?c1-data ?c1-rel)
  (child3-boundary ?parent3 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
  ?f1<- (child3-boundary ?parent3 ?child3&~?child2&~?child1 ?c1-data ?c3-rel)
  (test (or (eq ?c3-rel ?c2-rel)
            (eq ?c3-rel ?c1-rel)))

  =>
  (retract ?f1)
)

```

```

(defrule clear-3child-2mid
  (declare (salience -1))
  ?f1<-(child3-boundary ?parent3 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child3-boundary ?parent3 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)

```

```

=>
(retract ?f1)
(retract ?f2)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Remove the duplicated boundary arrows.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule remove-3child-3boundary
  (child3-boundary ?parent3 ?child1 ?c1-data ?c1-rel)
  ?f2<- (child3-boundary ?parent3 ?child2&~?child1 ?c1-data ?c1-rel)
  ?f3<- (child3-boundary ?parent3 ?child3&~?child2&~?child1 ?c1-data ?c1-rel)
  =>
  (retract ?f2 ?f3)
)

(defrule remove-3child-2boundary
  (child3-boundary ?parent3 ?child1 ?c1-data ?c1-rel)
  ?f2<- (child3-boundary ?parent3 ?child2&~?child1 ?c1-data ?c1-rel)
  =>
  (retract ?f2)
)

;-----

(defrule rid-3child-2consists
  ?f1<- (child3-boundary ?parent3 ?child1 ?c1-data ?c1-rel)
  ?f2<- (child3-boundary ?parent3 ?child2&~?child1 ?c2-data&~?c1-data ?c2-rel)
  ?f3<- (child3-boundary ?parent3 ?child-p&~?child1&~?child2
           ?cp-data&~?c2-data&~?c1-data ?cp-rel&~?c1-rel&~?c2-rel)
  (consists-of-name ? ?cp-data ?c2-data)
  (consists-of-name ? ?cp-data ?c1-data)
  =>
  (retract ?f1 ?f2 ?f3)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule check a parent with 3 child diagrams to see if the
; parent boundary data are also a part of it child's boundary
; data.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule check-3child-parent
  (declare (salience -5))
  (parent3-boundary ?p-name ?p-data ?p-rel)

```

```

(not (consists-of-name ? ?p-data ?c-data))
(not (child3-boundary ?p-name ?child3 ?p-data ?c3-rel))

=>
(printout t "ERROR: Data inconsistency between parent activity
"?p-name " data "' ?p-rel "' " ?p-data " and its child
diagrams." crlf)
(assert (syntax-error-occurred))
)

(defrule check-3child-parent-consists
(declare (salience -6))
(parent3-boundary ?p-name ?p-data ?p-rel)
(consists-of-name ? ?p-data ?c-data)
(not (child3-boundary ?p-name ?child3 ?c-data ?c3-rel))
=>
  (printout t "ERROR: Data inconsistency between parent activity
"?p-name " data "' ?p-rel "' " ?p-data " and its child
diagrams." crlf)
  (assert (syntax-error-occurred))
  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule check if a parent with 3 child diagram nat
; some of them have the same boundary data element but
; with different icom relation.
; Then it is an icom ERROR.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule check-3child-icom
(declare (salience -5))
(parent3-boundary ?p-name ?p-data ?p-rel)
(child3-boundary ?p-name ?c-name ?p-data ?c-rel)
(test (neq ?p-rel ?c-rel))
=>
  (printout t "ERROR: icom inconsistency between activity " crlf)
  (printout t " "?p-name " and its child diagram " ?c-name"." crlf)
  (assert (syntax-error-occurred))
  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule checks if a child has some boundary data element
; but can't find the same data in its parent then inconsistency
; happened.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```



```

?f1<-(parent3-icom ?p3-name ?data1 control ?one)
?f2<-(parent3-icom ?p3-name ?data2 control ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (parent3-icom ?p3-name =(gensym) control ?total))
)

; -----

(defrule parent3-output-add
(declare (salience -3))
?f1<-(parent3-icom ?p3-name ?data1 output ?one)
?f2<-(parent3-icom ?p3-name ?data2 output ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (parent3-icom ?p3-name =(gensym) output ?total))
)

; -----

(defrule parent3-input-add
(declare (salience -3))
?f1<-(parent3-icom ?p3-name ?data1 input ?one)
?f2<-(parent3-icom ?p3-name ?data2 input ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (parent3-icom ?p3-name =(gensym) input ?total))
)

(defrule parent3-mech-add
(declare (salience -3))
?f1<-(parent3-icom ?p3-name ?data1 mech ?one)
?f2<-(parent3-icom ?p3-name ?data2 mech ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (parent3-icom ?p3-name =(gensym) mech ?total))
)

; -----

(defrule child3-icom-c

```

```

(declare (salience -2))
(child3-boundary ?c3-parent ?c3-name ?c3-data c)
=>
(assert (child3-icom ?c3-parent ?c3-data control 1))
)

(defrule child3-icom-o
(declare (salience -2))
(child3-boundary ?c3-parent ?c3-name ?c3-data o)
=>
(assert (child3-icom ?c3-parent ?c3-data output 1) )
)

(defrule child3-icom-i
(declare (salience -2))
(child3-boundary ?c3-parent ?c3-name ?c3-data i)
=>
(assert (child3-icom ?c3-parent ?c3-data input 1) )
)

(defrule child3-icom-m
(declare (salience -2))
(child3-boundary ?c3-parent ?c3-name ?c3-data m)
=>
(assert (child3-icom ?c3-parent ?c3-data mech 1) )
)

; -----

(defrule child3-control-add
(declare (salience -3))
?f1<-(child3-icom ?c3-parent ?data1 control ?one)
?f2<-(child3-icom ?c3-parent ?data2 control ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child3-icom ?c3-parent =(gensym) control ?total))
)

; -----

(defrule child3-output-add
(declare (salience -3))
?f1<-(child3-icom ?c3-parent ?data1 output ?one)
?f2<-(child3-icom ?c3-parent ?data2 output ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child3-icom ?c3-parent =(gensym) output ?total))
)

```

```

)

; -----

(defrule child3-input-add
  (declare (salience -3))
  ?f1<-(child3-icom ?c3-parent ?data1 input ?one)
  ?f2<-(child3-icom ?c3-parent ?data2 input ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (child3-icom ?c3-parent =(gensym) input ?total))
  )

; -----

(defrule child3-mech-add
  (declare (salience -3))
  ?f1<-(child3-icom ?c3-parent ?data1 mech ?one)
  ?f2<-(child3-icom ?c3-parent ?data2 mech ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (child3-icom ?c3-parent =(gensym) mech ?total))
  )

;*****
;          Check Parent with 3 child  boundary icom number consistancy
;-----

(defrule check-parent-3child-control-no-retract
  (declare (salience -6))
  (parent3-icom ?p3-name ? control ?p)
  (child3-icom ?p3-name ? control ?c)
  (test (!= ?p ?c))
  =>
  (if (> ?p ?c)
    then
      (bind ?pd (- ?p ?c))
      (printout t "WARNING, there might be an ERROR: The number of boundary controls" crlf)
      (printout t "   of parent activity "?p3-name" is "?pd" control(s) more than " crlf)
      (printout t "   its child activities." crlf)
      (printout t "   Are there 'consists of' data items at boundary?" crlf)
      (printout t "   Please recheck the syntax." crlf)
    else
      (bind ?cd (- ?c ?p))
      (printout t "WARNING, there might be an ERROR: The number of boundary controls" crlf)
      (printout t "   of the parent activity "?p3-name" is "?cd" control(s) less " crlf)
  )

```

```

    (printout t "   than its child boundary controls." crlf)
    (printout t "   Are there 'consists of' data items at boundary?" crlf)
    (printout t "   Please recheck the syntax." crlf)
  ) )

(defrule check-parent-3child-output
  (declare (salience -6))
  ?f1<-(parent3-icom ?p3-name ? output ?p)
  ?f2<-(child3-icom ?p3-name ? output ?c)
  (test (!= ?p ?c))
  =>
  (retract ?f1 ?f2)
  (if (> ?p ?c)
    then
      (bind ?pd (- ?p ?c))
      (printout t "WARNING, there might be an ERROR: The number of boundary outputs" crlf)
      (printout t "   of parent activity " ?p3-name " is " ?pd " output(s) more " crlf)
      (printout t "   than its child activities." crlf)
      (printout t "   Are there 'consists of' data items at boundary?" crlf)
      (printout t "   Please recheck the syntax." crlf)
    else
      (bind ?cd (- ?c ?p))
      (printout t "WARNING, there might be an ERROR: The number of   boundary outputs" crlf)
      (printout t "   of the parent activity " ?p3-name " is " ?cd " output(s) less " crlf)
      (printout t "   than its child boundary outputs." crlf)
      (printout t "   Are there 'consists of' data items at boundary?" crlf)
      (printout t "   Please recheck the syntax." crlf)
  ) )

(defrule check-parent-3child-input
  (declare (salience -6))
  ?f1<-(parent3-icom ?p3-name ? input ?p)
  ?f2<-(child3-icom ?p3-name ? input ?c)
  (test (!= ?p ?c))
  =>
  (retract ?f1 ?f2)
  (if (> ?p ?c)
    then
      (bind ?pd (- ?p ?c))
      (printout t "WARNING, there might be an ERROR: The number of boundary inputs" crlf)
      (printout t "   of parent activity " ?p3-name " is " ?pd " input(s) more " crlf)
      (printout t "   than its child activities." crlf)
      (printout t "   Are there 'consists of' data items at boundary?" crlf)
      (printout t "   Please recheck the syntax." crlf)
    else
      (bind ?cd (- ?c ?p))
      (printout t "WARNING, there might be an ERROR: The number of   boundary inputs" crlf)
      (printout t "   of the parent activity " ?p3-name " is " ?cd " input(s) less " crlf)
      (printout t "   than its child boundary inputs." crlf)
      (printout t "   Are there 'consists of' data items at boundary?" crlf)
      (printout t "   Please recheck the syntax." crlf)
  ) )

```


))

```
(defrule check-parent-3child-mech
  (declare (salience -6))
  ?f1<-(parent3-icom ?p3-name ? mech ?p)
  ?f2<-(child3-icom ?p3-name ? mech ?c)
  (test (≠ ?p ?c))
  =>
  (retract ?f1 ?f2)
  (if (> ?p ?c)
    then
    (bind ?pd (- ?p ?c))
    (printout t "WARNING, there might be an ERROR: The number of boundary mechanisms" crlf)
    (printout t "  of parent activity " ?p3-name " is " ?pd " mechanism(s) more " crlf)
    (printout t "  than its child activities." crlf)
    (printout t "  Are there 'consists of' data items at boundary?" crlf)
    (printout t "  Please recheck the syntax." crlf)
  else
    (bind ?cd (- ?c ?p))
    (printout t "WARNING, there might be an ERROR: The number of boundary mechanisms" crlf)
    (printout t "  of the parent activity " ?p3-name " is " ?cd " mechanism(s) less " crlf)
    (printout t "  its child child boundary mechanisms." crlf)
    (printout t "  Are there 'consists of' data items at boundary?" crlf)
    (printout t "  Please recheck the syntax." crlf)
  ) )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule will create the boundary facts for activities having
; 4 child diagrams.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(defrule parent-4child
  (declare (salience 100))
  ?pf1<- (act-has-child ?parent4 ?child1&~null)
  ?pf2<- (act-has-child ?parent4 ?child2&~?child1&~null)
  ?pf3<- (act-has-child ?parent4 ?child3&~?child2&~?child1&~null)
  ?pf4<- (act-has-child ?parent4
    ?child4&~?child3&~?child2&~?child1&~null)
  (not (act-has-child ?parent4
    ?child5&~?child4&~?child3&~?child2&~?child1&~null) )
  =>
  (retract ?pf1 ?pf2 ?pf3 ?pf4)
  (assert (parent4 ?parent4 ?child1 ?child2 ?child3 ?child4))
  )

(defrule parent4-boundary
  (parent4 ?parent4 ?child1 ?child2 ?child3 ?child4)
  (icom-tuple ?parent4 ?p-data ?p-rel ?)
  =>
  (assert (parent4-boundary ?parent4 ?p-data ?p-rel))
```

```

)

(defrule child4-boundary-child1
  (parent4 ?parent4 ?child1 ?child2 ?child3 ?child4)
  (icom-tuple ?child1 ?c1-data ?c1-rel ?)
  =>
  (assert (child4-boundary ?parent4 ?child1 ?c1-data ?c1-rel))
)

(defrule child4-boundary-child2
  (parent4 ?parent4 ?child1 ?child2 ?child3 ?child4)
  (icom-tuple ?child2 ?c2-data ?c2-rel ?)
  =>
  (assert (child4-boundary ?parent4 ?child2 ?c2-data ?c2-rel))
)

(defrule child4-boundary-child3
  (parent4 ?parent4 ?child1 ?child2 ?child3 ?child4)
  (icom-tuple ?child3 ?c3-data ?c3-rel ?)
  =>
  (assert (child4-boundary ?parent4 ?child3 ?c3-data ?c3-rel))
)

(defrule child4-boundary-child4
  (parent4 ?parent4 ?child1 ?child2 ?child3 ?child4)
  (icom-tuple ?child4 ?c4-data ?c4-rel ?)
  =>
  (assert (child4-boundary ?parent4 ?child4 ?c4-data ?c4-rel))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; These rules will clear the duplicated boundary facts in the facts
; created by the previous rule.
; CONDITION:
;   1. 3 activities out of 4 sharing a same data
;   2. 2 activities out of 4 sharing a same data
;   3. all 4 activities are sharing a same data
;      element but with different icom code
; Condition 3 is not likely to happen, so it is not implemented
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule clear-4child-3mid
  ?f1<-(child4-boundary ?parent4 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child4-boundary ?parent4 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
  ?f3<-(child4-boundary ?parent4 ?child3&~?child2&~?child1 ?c1-data
              ?c3-rel&~?c2-rel&~?c1-rel)

  =>
  (retract ?f1)

```

```

(retract ?f2)
(retract ?f3)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; If a intermediate arrow is the input of one box but also the
; output and input of another two boxes. It must be removed before
; the arrow between the other boxes been removed.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule clear-4child-2mid-1
  (child4-boundary ?parent4 ?child1 ?c1-data ?c1-rel)
  (child4-boundary ?parent4 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
  ?f1<-(child4-boundary ?parent4 ?child3&~?child2&~?child1 ?c1-data ?c3-rel)
  (test (or (eq ?c3-rel ?c2-rel)
            (eq ?c3-rel ?c1-rel)) )
=>
(retract ?f1)
)

(defrule clear-4child-2mid
  (declare (salience -1))
  ?f1<-(child4-boundary ?parent4 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child4-boundary ?parent4 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
  =>
  (retract ?f1)
  (retract ?f2)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Remove the duplicated boundary arrows for parents with
; with 4 child diagrams. Consider that at most 3 child
; out of 4 might use the same data.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule remove-4child-3boundary
  (child4-boundary ?parent4 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child4-boundary ?parent4 ?child2&~?child1 ?c1-data ?c1-rel)
  ?f3<-(child4-boundary ?parent4 ?child3&~?child2&~?child1 ?c1-data ?c1-rel)
  =>
  (retract ?f2 ?f3)
)

(defrule remove-4child-2boundary
  (child4-boundary ?parent4 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child4-boundary ?parent4 ?child2&~?child1 ?c1-data ?c1-rel)

```

```

=>
(retract ?f2)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule will get rid of consists of intermediate data relations
; that a data has 3 subcomponents.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule rid-4child-3consists
  ?f1<-(child4-boundary ?parent4 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child4-boundary ?parent4 ?child2&~?child1 ?c2-data&~?c1-data ?c2-rel)
  ?f3<-(child4-boundary ?parent4 ?child3&~?child2&~?child1 ?c3-data&~?c2-data&~?c1-data ?c3-rel)
  ?f4<-(child4-boundary ?parent4 ?child-p&~?child3&~?child2&~?child1
    ?cp-data&~?c3-data&~?c2-data&~?c1-data ?cp-rel&~?c3-rel&~?c2-rel&~?c1-rel)
    (consists-of-name ? ?cp-data ?c3-data)
    (consists-of-name ? ?cp-data ?c2-data)
    (consists-of-name ? ?cp-data ?c1-data)
  =>
  (retract ?f1)
  (retract ?f2)
  (retract ?f3)
  (retract ?f4)
  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule should fired later than -3consists; since if
; 2 of 3 consists facts are retracted, the remaining one will not
; be matched to be retracted.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule rid-4child-2consists
  (declare (salience -1))
  ?f1<-(child4-boundary ?parent4 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child4-boundary ?parent4 ?child2&~?child1 ?c2-data&~?c1-data ?c2-rel)
  ?f3<-(child4-boundary ?parent4 ?child-p&~?child2&~?child1
    ?cp-data&~?c2-data&~?c1-data ?cp-rel&~?c2-rel&~?c1-rel)
    (consists-of-name ? ?cp-data ?c2-data)
    (consists-of-name ? ?cp-data ?c1-data)
  =>
  (retract ?f1 ?f2 ?f3)
  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule check a parent activity with 4 child diagram to see if
; there are any boundary data belonging to the parent but not a part of
; the child diagrams.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule check-4child-parent

```

```

(declare (salience -5))
(parent4-boundary ?p-name ?p-data ?p-rel)
(not (consists-of-name ? ?p-data ?c-data))
(not (child4-boundary ?p-name ?child4 ?p-data ?c4-rel))
=>
  (printout t "ERROR: Data inconsistency between parent activity
    "?p-name " data '" ?p-rel "' " ?p-data " and its child
    diagrams." crlf)
  (assert (syntax-error-occurred))
  )

(defrule check-4child-parent-consists
  (declare (salience -6))
  (parent4-boundary ?p-name ?p-data ?p-rel)
  (consists-of-name ? ?p-data ?c-data)
  (not (child4-boundary ?p-name ?child4 ?c-data ?c4-rel))
  =>
    (printout t "ERROR: Data inconsistency between parent activity
      "?p-name " data '" ?p-rel "' " ?p-data " and its child
      diagrams." crlf)
    (assert (syntax-error-occurred))
    )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule checks if a parent with 4 child diagrams that
; some of them have the same boundary data element but
; with different icom relation in contrast with their parent.
; Then it is an icom ERROR.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule check-4child-icom
  (declare (salience -5))
  (parent4-boundary ?p-name ?p-data ?p-rel)
  (child4-boundary ?p-name ?c-name ?p-data ?c-rel)
  (test (neq ?p-rel ?c-rel))
  =>
    (printout t "ERROR: icom inconsistency between activity "
      ?p-name " and its child diagram "
      ?c-name "." crlf)
    (assert (syntax-error-occurred))
    )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule checks if a parent have 4 child, and there is some
; boundary data element in the child diagrams
; but can't find the same data in their parent then inconsistency
; happened.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule check-4child-child
  (declare (salience -5))

```

```

(child4-boundary ?p-name ?c-name ?c-data ?c-rel)
(not (consists-of-name ? ?p-data ?c-data))
(not (parent4-boundary ?p-name ?c-data ?p-rel))
=>
  (printout t "ERROR: Data inconsistency between child activity
    " ?c-name " data '" ?c-rel "' " ?c-data " and its
    parent." crlf)
  (assert (syntax-error-occurred))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                               Parent with 4 child diagrams
;                               -----
; The initial icom number was build up by this rule,
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule parent4-icom-c
  (declare (salience -2))
  (parent4-boundary ?p4-name ?p4-data c)
  =>
  (assert (parent4-icom ?p4-name ?p4-data control 1))
)

(defrule parent4-icom-o
  (declare (salience -2))
  (parent4-boundary ?p4-name ?p4-data o)
  =>
  (assert (parent4-icom ?p4-name ?p4-data output 1))
)

(defrule parent4-icom-i
  (declare (salience -2))
  (parent4-boundary ?p4-name ?p4-data i)
  =>
  (assert (parent4-icom ?p4-name ?p4-data input 1))
)

(defrule parent4-icom-m
  (declare (salience -2))
  (parent4-boundary ?p4-name ?p4-data m)
  =>
  (assert (parent4-icom ?p4-name ?p4-data mech 1))
)

; -----

(defrule parent4-control-add
  (declare (salience -3))
  ?f1<-(parent4-icom ?p4-name ?data1 control ?one)
  ?f2<-(parent4-icom ?p4-name ?data2 control ?n)
  (test (neq ?data1 ?data2))

```

```

=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (parent4-icom ?p4-name =(gensym) control ?total))
)

; -----

(defrule parent4-output-add
(declare (salience -3))
?f1<-(parent4-icom ?p4-name ?data1 output ?one)
?f2<-(parent4-icom ?p4-name ?data2 output ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (parent4-icom ?p4-name =(gensym) output ?total))
)

; -----

(defrule parent4-input-add
(declare (salience -3))
?f1<-(parent4-icom ?p4-name ?data1 input ?one)
?f2<-(parent4-icom ?p4-name ?data2 input ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (parent4-icom ?p4-name =(gensym) input ?total))
)

; -----

(defrule parent4-mech-add
(declare (salience -3))
?f1<-(parent4-icom ?p4-name ?data1 ?mech ?one)
?f2<-(parent4-icom ?p4-name ?data2 ?mech ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (parent4-icom ?p4-name =(gensym) mech ?total))
)

; -----

(defrule child4-icom-c
(declare (salience -2))
(child4-boundary ?c4-parent ?c4-name ?c4-data c)

```

```

=>
(assert (child4-icom ?c4-parent ?c4-data control 1))
)

(defrule child4-icom-o
  (declare (salience -2))
  (child4-boundary ?c4-parent ?c4-name ?c4-data o)
  =>
  (assert (child4-icom ?c4-parent ?c4-data output 1) )
  )

(defrule child4-icom-i
  (declare (salience -2))
  (child4-boundary ?c4-parent ?c4-name ?c4-data i)
  =>
  (assert (child4-icom ?c4-parent ?c4-data input 1) )
  )

(defrule child4-icom-m
  (declare (salience -2))
  (child4-boundary ?c4-parent ?c4-name ?c4-data m)
  =>
  (assert (child4-icom ?c4-parent ?c4-data mech 1) )
  )

; -----

(defrule child4-control-add
  (declare (salience -3))
  ?f1<-(child4-icom ?c4-parent ?data1 control ?one)
  ?f2<-(child4-icom ?c4-parent ?data2 control ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (child4-icom ?c4-parent =(gensym) control ?total))
  )

; -----

(defrule child4-output-add
  (declare (salience -3))
  ?f1<-(child4-icom ?c4-parent ?data1 output ?one)
  ?f2<-(child4-icom ?c4-parent ?data2 output ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (child4-icom ?c4-parent =(gensym) output ?total))
  )

```



```

)

; -----

(defrule child4-input-add
  (declare (salience -3))
  ?f1<-(child4-icom ?c4-parent ?data1 input ?one)
  ?f2<-(child4-icom ?c4-parent ?data2 input ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (child4-icom ?c4-parent =(gensym) input ?total))
  )

; -----

(defrule child4-mech-add
  (declare (salience -3))
  ?f1<-(child4-icom ?c4-parent ?data1 mech ?one)
  ?f2<-(child4-icom ?c4-parent ?data2 mech ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (child4-icom ?c4-parent =(gensym) mech ?total))
  )

;*****
;          Check Parent with 4 child  boundary icom number consistancy
;-----

(defrule check-parent-4child-control
  (declare (salience -6))
  ?f1<-(parent4-icom ?p4-name ? control ?p)
  ?f2<-(child4-icom ?p4-name ? control ?c)
  (test (!= ?p ?c))
  =>
  (retract ?f1 ?f2)
  (if (> ?p ?c)
    then
    (bind ?pd (- ?p ?c))
    (printout t "WARNING, there might be an ERROR: The number of boundary controls" crlf)
    (printout t "  of parent activity "?p4-name" is "?pd" control(s) more than " crlf)
    (printout t "  its child activities." crlf)
    (printout t "  Are there 'consists of' data items at boundary?" crlf)
    (printout t "  Please recheck the syntax." crlf)
  else
    (bind ?cd (- ?c ?p))
  )

```

```

(printout t "WARNING, there might be an ERROR: The number of  boundary controls" crlf)
(printout t "  of the parent activity "?p4-name" is "?cd" control(s) less  " crlf)
(printout t "  than its child boundary controls." crlf)
(printout t "  Are there ''consists of'' data items at boundary?" crlf)
(printout t "  Please recheck the syntax." crlf)
) )

(defrule check-parent-4child-output
(declare (salience -6))
?f1<-(parent4-icom ?p4-name ? output ?p)
?f2<-(child4-icom ?p4-name ? output ?c)
(test (!= ?p ?c))
=>
(retract ?f1 ?f2)
(if (> ?p ?c)
then
(bind ?pd (- ?p ?c))
(printout t "WARNING, there might be an ERROR: The number of boundary outputs" crlf)
(printout t "  of parent activity " ?p4-name " is " ?pd " output(s) more  " crlf)
(printout t "  than its child activities." crlf)
(printout t "  Are there ''consists of'' data items at boundary?" crlf)
(printout t "  Please recheck the syntax." crlf)
else
(bind ?cd (- ?c ?p))
(printout t "WARNING, there might be an ERROR: The number of  boundary outputs" crlf)
(printout t "  of the parent activity " ?p4-name " is " ?cd " output(s) less  " crlf)
(printout t "  than its child boundary outputs." crlf)
(printout t "  Are there ''consists of'' data items at boundary?" crlf)
(printout t "  Please recheck the syntax." crlf)
) )

(defrule check-parent-4child-input
(declare (salience -6))
?f1<-(parent4-icom ?p4-name ? input ?p)
?f2<-(child4-icom ?p4-name ? input ?c)
(test (!= ?p ?c))
=>
(retract ?f1 ?f2)
(if (> ?p ?c)
then
(bind ?pd (- ?p ?c))
(printout t "WARNING, there might be an ERROR: The number of boundary inputs" crlf)
(printout t "  or parent activity " ?p4-name " is " ?pd " input(s) more  " crlf)
(printout t "  than its child activities." crlf)
(printout t "  Are there ''consists of'' data items at boundary?" crlf)
(printout t "  Please recheck the syntax." crlf)
else
(bind ?cd (- ?c ?p))
(printout t "WARNING, there might be an ERROR: The number of  boundary inputs" crlf)
(printout t "  of the parent activity " ?p4-name " is " ?cd " input(s) less  " crlf)

```

```

(printout t "   than its child boundary inputs." crlf)
(printout t "   Are there 'consists of' data items at boundary?" crlf)
(printout t "   Please recheck the syntax." crlf)
) )

```

```

(defrule check-parent-4child-mech
  (declare (salience -6))
  ?f1<-(parent4-icom ?p4-name ? mech ?p)
  ?f2<-(child4-icom ?p4-name ? mech ?c)
  (test (!= ?p ?c))
  =>
  (retract ?f1 ?f2)
  (if (> ?p ?c)
    then
    (bind ?pd (- ?p ?c))
    (printout t "WARNING, there might be an ERROR: The number of boundary mechanisms" crlf)
    (printout t "   of parent activity " ?p4-name " is " ?pd " mechanism(s) more " crlf)
    (printout t "   than its child activities." crlf)
    (printout t "   Are there 'consists of' data items at boundary?" crlf)
    (printout t "   Please recheck the syntax." crlf)
    else
    (bind ?cd (- ?c ?p))
    (printout t "WARNING, there might be an ERROR: The number of boundary mechanisms" crlf)
    (printout t "   of the parent activity " ?p4-name " is " ?cd " mechanism(s) less " crlf)
    (printout t "   its child child boundary mechanisms." crlf)
    (printout t "   Are there 'consists of' data items at boundary?" crlf)
    (printout t "   Please recheck the syntax." crlf)
  ) )

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule will create the boundary facts for activities having
; 5 child diagrams.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defrule parent-5child
  (declare (salience 100))
  ?f1<-(act-has-child ?parent5 ?child1&~null)
  ?f2<-(act-has-child ?parent5 ?child2&~?child1&~null)
  ?f3<-(act-has-child ?parent5 ?child3&~?child2&~?child1&~null)
  ?f4<-(act-has-child ?parent5
            ?child4&~?child3&~?child2&~?child1&~null)
  ?f5<-(act-has-child ?parent5
            ?child5&~?child4&~?child3&~?child2&~?child1&~null)
  (not (act-has-child ?parent5

```

```

        ?child6&~?child5&~?child4&~?child3&~?child2&~?child1&~null) )
=>
  (retract ?f1 ?f2 ?f3 ?f4 ?f5)
  (assert (parent5 ?parent5 ?child1 ?child2 ?child3 ?child4 ?child5))
)

(defrule parent5-boundary
  (parent5 ?parent5 ?child1 ?child2 ?child3 ?child4 ?child5)
  (icom-tuple ?parent5 ?p-data ?p-rel ?)
=>
  (assert (parent5-boundary ?parent5 ?p-data ?p-rel))
)

(defrule child5-boundary-child1
  (parent5 ?parent5 ?child1 ?child2 ?child3 ?child4 ?child5)
  (icom-tuple ?child1 ?c1-data ?c1-rel ?)
=>
  (assert (child5-boundary ?parent5 ?child1 ?c1-data ?c1-rel))
)

(defrule child5-boundary-child2
  (parent5 ?parent5 ?child1 ?child2 ?child3 ?child4 ?child5)
  (icom-tuple ?child2 ?c2-data ?c2-rel ?)
=>
  (assert (child5-boundary ?parent5 ?child2 ?c2-data ?c2-rel))
)

(defrule child5-boundary-child3
  (parent5 ?parent5 ?child1 ?child2 ?child3 ?child4 ?child5)
  (icom-tuple ?child3 ?c3-data ?c3-rel ?)
=>
  (assert (child5-boundary ?parent5 ?child3 ?c3-data ?c3-rel))
)

(defrule child5-boundary-child4
  (parent5 ?parent5 ?child1 ?child2 ?child3 ?child4 ?child5)
  (icom-tuple ?child4 ?c4-data ?c4-rel ?)
=>
  (assert (child5-boundary ?parent5 ?child4 ?c4-data ?c4-rel))
)

(defrule child5-boundary-child5
  (parent5 ?parent5 ?child1 ?child2 ?child3 ?child4 ?child5)
  (icom-tuple ?child5 ?c5-data ?c5-rel ?)
=>
  (assert (child5-boundary ?parent5 ?child5 ?c5-data ?c5-rel))
)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```
; These rules will clear the duplicated boundary facts in the facts
; created by the previous rule.
; CONDITION:
;     1. 4 activities out of 5 sharing a same data
;     2. 3 activities out of 5 sharing a same data
;     3. 2 activities out of 5 sharing a same data
;     4. all 5 activities are sharing a same data
;         element but with different icom code
; Condition 1 and 4 is not likely to happen, so it is not implemented
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule clear-5child-3mid
?f1<-(child5-boundary ?parent5 ?child1 ?c1-data ?c1-rel)
?f2<-(child5-boundary ?parent5 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
?f3<-(child5-boundary ?parent5 ?child3&~?child2&~?child1 ?c1-data
      ?c3-rel&~?c2-rel&~?c1-rel)
=>
(retract ?f1)
(retract ?f2)
(retract ?f3)
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; If a intermediate arrow is the input of one box but also the
; output and input of another two boxes. It must be removed before
; the arrow between the other boxes been removed.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule clear-5child-2mid-1
(child5-boundary ?paernt5 ?child1 ?c1-data ?c1-rel)
(child5-boundary ?parent5 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
?f1<-(child5-boundary ?parent5 ?child3&~?child2&~?child1 ?c1-data ?c3-rel)
(test (or (eq ?c3-rel ?c2-rel)
          (eq ?c3-rel ?c1-rel))) )
=>
(retract ?f1)
)
;
;
(defrule clear-5child-2mid
(declare (salience -1))
?f1<-(child5-boundary ?parent5 ?child1 ?c1-data ?c1-rel)
?f2<-(child5-boundary ?parent5 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
=>
(retract ?f1)
(retract ?f2)
)
;
;
;
```

```

; Remove the duplicated boundary arrows for parents with
; with 5 child diagrams. Consider that at most 3 child
; out of 5 might use the same data.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule remove-5child-3boundary
  (child5-boundary ?parent5 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child5-boundary ?parent5 ?child2&~?child1 ?c1-data ?c1-rel)
  ?f3<-(child5-boundary ?parent5 ?child3&~?child2&~?child1 ?c1-data ?c1-rel)
  =>
  (retract ?f2 ?f3)
)

(defrule remove-5child-2boundary
  (child5-boundary ?parent5 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child5-boundary ?parent5 ?child2&~?child1 ?c1-data ?c1-rel)
  =>
  (retract ?f2)
)

(defrule rid-5child-3consists
  ?f1<-(child5-boundary ?parent5 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child5-boundary ?parent5 ?child2&~?child1 ?c2-data&~?c1-data ?c2-rel)
  ?f3<-(child5-boundary ?parent5 ?child3&~?child2&~?child1
        ?c3-data&~?c2-data&~?c1-data ?c3-rel)
  ?f4<-(child5-boundary ?parent5 ?child-p&~?child3&~?child2&~?child1
        ?cp-data&~?c3-data&~?c2-data&~?c1-data ?cp-rel&~?c3-rel&~?c2-rel&~?c1-rel)
  (consists-of-name ? ?cp-data ?c3-data)
  (consists-of-name ? ?cp-data ?c2-data)
  (consists-of-name ? ?cp-data ?c1-data)
  =>
  (retract ?f1)
  (retract ?f2)
  (retract ?f3)
  (retract ?f4)
)

(defrule rid-5child-2consists
  (declare (salience -1))
  ?f1<-(child5-boundary ?parent5 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child5-boundary ?parent5 ?child2&~?child1 ?c2-data&~?c1-data ?c2-rel)
  ?f3<-(child5-boundary ?parent5 ?child-p&~?child2&~?child1 ?cp-data&~?c2-data&~?c1-data
        ?cp-rel&~?c2-rel&~?c1-rel)
  (consists-of-name ? ?cp-data ?c2-data)
  (consists-of-name ? ?cp-data ?c1-data)
  =>
  (retract ?f1 ?f2 ?f3)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```
; This rule check a parent activity with 5 child diagram to see if
; there are any boundary data belonging to the parent but not a part of
; the child diagrams.
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
(defrule check-5child-parent
  (declare (salience -5))
  (parent5-boundary ?p-name ?p-data ?p-rel)
  (not (consists-of-name ? ?p-data ?c-data))
  (not (child5-boundary ?p-name ?child5 ?p-data ?c5-rel))
  =>
    (printout t "ERROR: Data inconsistency between parent activity
    "?p-name " data "" ?p-rel "" " ?p-data " and its child
    diagrams." crlf)
    (assert (syntax-error-occurred))
  )
```

```
(defrule check-5child-parent-consists
  (declare (salience -6))
  (parent5-boundary ?p-name ?p-data ?p-rel)
  (consists-of-name ? ?p-data ?c-data)
  (not (child5-boundary ?p-name ?child5 ?c-data ?c5-rel))
  =>
    (printout t "ERROR: Data inconsistency between parent activity
    "?p-name " data "" ?p-rel "" " ?p-data " and its child
    diagrams." crlf)
    (assert (syntax-error-occurred))
  )
```

```
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; This rule checks if a parent with 5 child diagrams that
; some of them have the same boundary data element but
; with different icom relation in contrast with their parent.
; Then it is an icom ERROR.
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
(defrule check-5child-icom
  (declare (salience -5))
  (parent5-boundary ?p-name ?p-data ?p-rel)
  (child5-boundary ?p-name ?c-name ?p-data ?c-rel)
  (test (neq ?p-rel ?c-rel))
  =>
    (printout t "ERROR: icom inconsistency between activity "
    ?p-name " and its child diagram " ?c-name "." crlf)
    (assert (syntax-error-occurred))
  )
```

```
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
; This rule checks if a parent have 5 child, and there is some
```



```

)

; -----

(defrule parent5-control-add
  (declare (salience -3))
  ?f1<-(parent5-icom ?p5-name ?data1 control ?one)
  ?f2<-(parent5-icom ?p5-name ?data2 control ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent5-icom ?p5-name =(gensym) control ?total))
)

; -----

(defrule parent5-output-add
  (declare (salience -3))
  ?f1<-(parent5-icom ?p5-name ?data1 output ?one)
  ?f2<-(parent5-icom ?p5-name ?data2 output ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent5-icom ?p5-name =(gensym) output ?total))
)

; -----

(defrule parent5-input-add
  (declare (salience -3))
  ?f1<-(parent5-icom ?p5-name ?data1 input ?one)
  ?f2<-(parent5-icom ?p5-name ?data2 input ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent5-icom ?p5-name =(gensym) input ?total))
)

(defrule parent5-mech-add
  (declare (salience -3))
  ?f1<-(parent5-icom ?p5-name ?data1 mech ?one)
  ?f2<-(parent5-icom ?p5-name ?data2 mech ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))

```

```

    (assert (parent5-icom ?p5-name =(gensym) mech ?total))
  )

; -----

(defrule child5-icom-c
  (declare (salience -2))
  (child5-boundary ?c5-parent ?p5-name ?c5-data c)
  =>
  (assert (child5-icom ?c5-parent ?c5-data control 1))
  )

(defrule child5-icom-o
  (declare (salience -2))
  (child5-boundary ?c5-parent ?c5-name ?c5-data o)
  =>
  (assert (child5-icom ?c5-parent ?c5-data output 1) )
  )

(defrule child5-icom-i
  (declare (salience -2))
  (child5-boundary ?c5-parent ?c5-name ?c5-data i)
  =>
  (assert (child5-icom ?c5-parent ?c5-data input 1) )
  )

(defrule child5-icom-m
  (declare (salience -2))
  (child5-boundary ?c5-parent ?c5-name ?c5-data m)
  =>
  (assert (child5-icom ?c5-parent ?c5-data mech 1) )
  )

; -----

(defrule child5-control-add
  (declare (salience -3))
  ?f1<-(child5-icom ?c5-parent ?data1 control ?one)
  ?f2<-(child5-icom ?c5-parent ?data2 control ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (child5-icom ?c5-parent =(gensym) control ?total))
  )

; -----

(defrule child5-output-add

```

```

(declare (salience -3))
?f1<-(child5-icom ?c5-parent ?data1 output ?one)
?f2<-(child5-icom ?c5-parent ?data2 output ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child5-icom ?c5-parent =(gensym) output ?total))
)

; -----

(defrule child5-input-add
(declare (salience -3))
?f1<-(child5-icom ?c5-parent ?data1 input ?one)
?f2<-(child5-icom ?c5-parent ?data2 input ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child5-icom ?c5-parent =(gensym) input ?total))
)

; -----

(defrule child5-mech-add
(declare (salience -3))
?f1<-(child5-icom ?c5-parent ?data1 mech ?one)
?f2<-(child5-icom ?c5-parent ?data2 mech ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child5-icom ?c5-parent =(gensym) mech ?total))
)

;*****
;          Check Parent with 5 child boundary icom number consistency
;-----

(defrule check-parent-5child-control
(declare (salience -6))
(parent5-icom ?p5-name ? control ?p)
(child5-icom ?p5-name ? control ?c)
(test (!= ?p ?c))
=>
(if (> ?p ?c)
then
(bind ?pd (- ?p ?c))
(printout t "WARNING, there might be an ERROR: The number of boundary controls" crlf)
(printout t "   of parent activity "?p5-name" is "?pd" control(s) more than " crlf)
(printout t "   its child activities." crlf)
)
)

```

```

(printout t " Are there 'consists of' data items at boundary?" crlf)
(printout t " Please recheck the syntax." crlf)
else
  (bind ?cd (- ?c ?p))
  (printout t "WARNING, there might be an ERROR: The number of boundary controls" crlf)
  (printout t " of the parent activity "?p5-name" is "?cd" control(s) less " crlf)
  (printout t " than its child boundary controls." crlf)
  (printout t " Are there 'consists of' data items at boundary?" crlf)
  (printout t " Please recheck the syntax." crlf)
) )

```

```

(defrule check-parent-5child-output
  (declare (salience -6))
  (parent5-icom ?p5-name ? output ?p)
  (child5-icom ?p5-name ? output ?c)
  (test (!= ?p ?c))
  =>
  (if (> ?p ?c)
    then
      (bind ?pd (- ?p ?c))
      (printout t "WARNING, there might be an ERROR: The number of boundary outputs" crlf)
      (printout t " of parent activity " ?p5-name " is " ?pd " output(s) more " crlf)
      (printout t " than its child activities." crlf)
      (printout t " Are there 'consists of' data items at boundary?" crlf)
      (printout t " Please recheck the syntax." crlf)
    else
      (bind ?cd (- ?c ?p))
      (printout t "WARNING, there might be an ERROR: The number of boundary outputs" crlf)
      (printout t " of the parent activity " ?p5-name " is " ?cd " output(s) less " crlf)
      (printout t " than its child boundary outputs." crlf)
      (printout t " Are there 'consists of' data items at boundary?" crlf)
      (printout t " Please recheck the syntax." crlf)
  ) )

```

```

(defrule check-parent-5child-input
  (declare (salience -6))
  (parent5-icom ?p5-name ? input ?p)
  (child5-icom ?p5-name ? input ?c)
  (test (!= ?p ?c))
  =>
  (if (> ?p ?c)
    then
      (bind ?pd (- ?p ?c))
      (printout t "WARNING, there might be an ERROR: The number of boundary inputs" crlf)
      (printout t " of parent activity " ?p5-name " is " ?pd " input(s) more " crlf)
      (printout t " than its child activities." crlf)
      (printout t " Are there 'consists of' data items at boundary?" crlf)
      (printout t " Please recheck the syntax." crlf)
    )
  )

```

```

else
  (bind ?cd (- ?c ?p))
  (printout t "WARNING, there might be an ERROR: The number of boundary inputs" crlf)
  (printout t " of the parent activity " ?p5-name " is " ?cd " input(s) less " crlf)
  (printout t " than its child boundary inputs." crlf)
  (printout t " Are there 'consists of' data items at boundary?" crlf)
  (printout t " Please recheck the syntax." crlf)
) )

(defrule check-parent-5child-mech
  (declare (salience -6))
  (parent5-icom ?p5-name ? mech ?p)
  (child5-icom ?p5-name ? mech ?c)
  (test (≠ ?p ?c))
  =>
  (if (> ?p ?c)
    then
      (bind ?pd (- ?p ?c))
      (printout t "WARNING, there might be an ERROR: The number of boundary mechanisms" crlf)
      (printout t " of parent activity " ?p5-name " is " ?pd " mechanism(s) more " crlf)
      (printout t " than its child activities." crlf)
      (printout t " Are there 'consists of' data items at boundary?" crlf)
      (printout t " Please recheck the syntax." crlf)
    else
      (bind ?cd (- ?c ?p))
      (printout t "WARNING, there might be an ERROR: The number of boundary mechanisms" crlf)
      (printout t " of the parent activity " ?p5-name " is " ?cd " mechnaism(s) less " crlf)
      (printout t " its child child boundary mechanisms." crlf)
      (printout t " Are there 'consists of' data items at boundary?" crlf)
      (printout t " Please recheck the syntax." crlf)
  ) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule will create the boundary facts for activities having
; 6 child diagrams.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule parent-6child
  (declare (salience 100))
  ?f1<-(act-has-child ?parent6 ?child1&~null)
  ?f2<-(act-has-child ?parent6 ?child2&~?child1&~null)
  ?f3<-(act-has-child ?parent6 ?child3&~?child2&~?child1&~null)
  ?f4<-(act-has-child ?parent6
           ?child4&~?child3&~?child2&~?child1&~null)
  ?f5<-(act-has-child ?parent6
           ?child5&~?child4&~?child3&~?child2&~?child1&~null)
  ?f6<-(act-has-child ?parent6

```

```

        ?child6&~?child5&~?child4&~?child3&~?child2&~?child1&~null)
(not (act-has-child ?parent6
    ?child7&~?child6&~?child5&~?child4&~?child3&~?child2&~?child1&~null) )

=>
(retract ?f1 ?f2 ?f3 ?f4 ?f5 ?f6)
(assert (parent6 ?parent6 ?child1 ?child2 ?child3 ?child4
    ?child5 ?child6))
)

(defrule parent6-boundary
  (parent6 ?parent6 ?child1 ?child2 ?child3 ?child4 ?child5 ?child6)
  (icom-tuple ?parent6 ?p-data ?p-rel ?)
  =>
  (assert (parent6-boundary ?parent6 ?p-data ?p-rel))
)

(defrule child6-boundary-child1
  (parent6 ?parent6 ?child1 ?child2 ?child3 ?child4 ?child5 ?child6)
  (icom-tuple ?child1 ?c1-data ?c1-rel ?)
  =>
  (assert (child6-boundary ?parent6 ?child1 ?c1-data ?c1-rel))
)

(defrule child6-boundary-child2
  (parent6 ?parent6 ?child1 ?child2 ?child3 ?child4 ?child5 ?child6)
  (icom-tuple ?child2 ?c2-data ?c2-rel ?)
  =>
  (assert (child6-boundary ?parent6 ?child2 ?c2-data ?c2-rel))
)

(defrule child6-boundary-child3
  (parent6 ?parent6 ?child1 ?child2 ?child3 ?child4 ?child5 ?child6)
  (icom-tuple ?child3 ?c3-data ?c3-rel ?)
  =>
  (assert (child6-boundary ?parent6 ?child3 ?c3-data ?c3-rel))
)

(defrule child6-boundary-child4
  (parent6 ?parent6 ?child1 ?child2 ?child3 ?child4 ?child5 ?child6)
  (icom-tuple ?child4 ?c4-data ?c4-rel ?)
  =>
  (assert (child6-boundary ?parent6 ?child4 ?c4-data ?c4-rel))
)

(defrule child6-boundary-child5
  (parent6 ?parent6 ?child1 ?child2 ?child3 ?child4 ?child5 ?child6)
  (icom-tuple ?child5 ?c5-data ?c5-rel ?)
  =>
  (assert (child6-boundary ?parent6 ?child5 ?c5-data ?c5-rel))
)

```

```

)

(defrule child6-boundary-child6
  (parent6 ?parent6 ?child1 ?child2 ?child3 ?child4 ?child5 ?child6)
  (icom-tuple ?child6 ?c6-data ?c6-rel ?)
  =>
    (assert (child6-boundary ?parent6 ?child6 ?c6-data ?c6-rel))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; These rules will clear the duplicated boundary facts in the facts
; created by the previous rule.
; CONDITION:
;   1. 5 activities out of 6 sharing a same data
;   2. 4 activities out of 6 sharing a same data
;   3. 3 activities out of 6 sharing a same data
;   4. 2 activities out of 6 sharing a same data
;   5. all 6 activities are sharing a same data
;      element but with different icom code
; Condition 1 and 5 is not likely to happen, so it is not implemented
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule clear-6child-4mid
  ?f1<-(child6-boundary ?parent6 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child6-boundary ?parent6 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
  ?f3<-(child6-boundary ?parent6 ?child3&~?child2&~?child1 ?c1-data
              ?c3-rel&~?c2-rel&~?c1-rel)
  ?f4<-(child6-boundary ?parent6 ?child4&~?child3&~?child2&~?child1 ?c1-data ?c4-rel)
    (test (or (and (neq ?c4-rel ?c1-rel)
                  (neq ?c4-rel ?c2-rel)
                  (eq ?c4-rel ?c3-rel) )
              (and (eq ?c4-rel ?c1-rel)
                  (neq ?c4-rel ?c2-rel)
                  (neq ?c4-rel ?c3-rel) )
              (and (neq ?c4-rel ?c1-rel)
                  (eq ?c4-rel ?c2-rel)
                  (neq ?c4-rel ?c3-rel) ) ) )
  =>
    (retract ?f1)
    (retract ?f2)
    (retract ?f3)
    (retract ?f4)
)

(defrule clear-6child-3mid
  ?f1<-(child6-boundary ?parent6 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child6-boundary ?parent6 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)

```

```

?f3<-(child6-boundary ?parent6 ?child3&~?child2&~?child1 ?c1-data
      ?c3-rel&~?c2-rel&~?c1-rel)

=>
(retract ?f1)
(retract ?f2)
(retract ?f3)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; If a intermediate arrow is the input of one box but also the
; output and input of another two boxes. It must be removed before
; the arrow between the other boxes been removed.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule clear-6child-2mid-1
  (child6-boundary ?parent6 ?child1 ?c1-data ?c1-rel)
  (child6-boundary ?parent6 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
  ?f1<-(child6-boundary ?parent6 ?child3&~?child2&~?child1 ?c1-data ?c3-rel)
  (test (or (eq ?c3-rel ?c2-rel)
            (eq ?c3-rel ?c1-rel)))
=>
(retract ?f1)
)

(defrule clear-6child-2mid
  (declare (salience -1))
  ?f1<-(child6-boundary ?parent6 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child6-boundary ?parent6 ?child2&~?child1 ?c1-data ?c2-rel&~?c1-rel)
  =>
  (retract ?f1)
  (retract ?f2)
)

(defrule remove-6child-4boundary
  (child6-boundary ?parent6 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child6-boundary ?parent6 ?child2&~?child1 ?c1-data ?c1-rel)
  ?f3<-(child6-boundary ?parent6 ?child3&~?child2&~?child1 ?c1-data ?c1-rel)
  ?f4<-(child6-boundary ?parent6 ?child4&~?child3&~?child2&~?child1 ?c1-data ?c1-rel)
  =>
  (retract ?f2 ?f3 ?f4)
)

(defrule remove-6child-3boundary
  (child6-boundary ?parent6 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child6-boundary ?parent6 ?child2&~?child1 ?c1-data ?c1-rel)

```



```

?f3<- (child6-boundary ?parent6 ?child3&~?child2&~?child1 ?c1-data ?c1-rel)
=>
(retract ?f2 ?f3)
)

(defrule remove-6child-2boundary
  (child6-boundary ?parent6 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child6-boundary ?parent6 ?child2&~?child1 ?c1-data ?c1-rel)
  =>
  (retract ?f2)
)

;-----
(defrule rid-6child-3consists
  ?f1<-(child6-boundary ?parent6 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child6-boundary ?parent6 ?child2&~?child1 ?c2-data&~?c1-data ?c2-rel)
  ?f3<-(child6-boundary ?parent6 ?child3&~?child2&~?child1
          ?c3-data&~?c2-data&~?c1-data ?c3-rel)
  ?f4<-(child6-boundary ?parent6 ?child-p&~?child3&~?child2&~?child1
          ?cp-data&~?c3-data&~?c2-data&~?c1-data ?cp-rel&~?c3-rel&~?c2-rel&~?c1-rel)
  (consists-of-name ? ?cp-data ?c3-data)
  (consists-of-name ? ?cp-data ?c2-data)
  (consists-of-name ? ?cp-data ?c1-data)
  =>
  (retract ?f1)
  (retract ?f2)
  (retract ?f3)
  (retract ?f4)
)

(defrule rid-6child-2consists
  (declare (salience -1))
  ?f1<-(child6-boundary ?parent6 ?child1 ?c1-data ?c1-rel)
  ?f2<-(child6-boundary ?parent6 ?child2&~?child1 ?c2-data&~?c1-data ?c2-rel)
  ?f3<-(child6-boundary ?parent6 ?child-p&~?child2&~?child1
          ?cp-data&~?c2-data&~?c1-data ?cp-rel&~?c2-rel&~?c1-rel)
  (consists-of-name ? ?cp-data ?c2-data)
  (consists-of-name ? ?cp-data ?c1-data)
  =>
  (retract ?f1 ?f2 ?f3)
)

;
; This rule check a parent activity with 6 child diagram to see if
; there are any boundary data belonging to the parent but not a part of
; the child diagrams.
;

```

```

(defrule check-6child-parent
  (declare (salience -5))
  (parent6-boundary ?p-name ?p-data ?p-rel)
  (not (consists-of-name ? ?p-data ?c-data))
  (not (child6-boundary ?p-name ?child6 ?p-data ?c6-rel))
=>
  (printout t "ERROR: Data inconsistency between parent activity
    "?p-name " data "' ?p-rel "' " ?p-data " and its child
    diagrams." crlf)
  (assert (syntax-error-occurred))
)

```

```

(defrule check-6child-parent-consists
  (declare (salience -6))
  (parent6-boundary ?p-name ?p-data ?p-rel)
  (consists-of-name ? ?p-data ?c-data)
  (not (child6-boundary ?p-name ?child6 ?c-data ?c6-rel))
=>
  (printout t "ERROR: Data inconsistency between parent activity
    "?p-name " data "' ?p-rel "' " ?p-data " and its child
    diagrams." crlf)
  (assert (syntax-error-occurred))
)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule checks if a parent with 6 child diagrams that
; some of them have the same boundary data element but
; with different icom relation in contrast with their parent.
; Then it is an icom ERROR.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defrule check-6child-icom
  (declare (salience -5))
  (parent6-boundary ?p-name ?p-data ?p-rel)
  (child6-boundary ?p-name ?c-name ?p-data ?c-rel)
  (test (neq ?p-rel ?c-rel))
=>
  (printout t "ERROR: icom inconsistency between activity "
    ?p-name " and its child diagram "
    ?c-name "." crlf)
  (assert (syntax-error-occurred))
)

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This rule checks if a parent have 6 child, and there is some
; boundary data element in the child diagrams
; but can't find the same data in their parent, then inconsistency
; happened.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defrule check-6child-child
  (declare (salience -5))
  (child6-boundary ?p-name ?c-name ?c-data ?c-rel)
  (not (consists-of-name ? ?p-data ?c-data))
  (not (parent6-boundary ?p-name ?c-data ?p-rel))
  =>
    (printout t "ERROR: Data inconsistency between child activity
      " ?c-name " data '" ?c-rel "' " ?c-data " and its
      parent." crlf)
    (assert (syntax-error-occurred))
  )

```

```

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
;                               Parent with 6 child diagrams
;                               -----
; The initial icom number was build up by this rule,
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

```

```

(defrule parent6-icom-c
  (declare (salience -2))
  (parent6-boundary ?p6-name ?p6-data c)
  =>
    (assert (parent6-icom ?p6-name ?p6-data control 1))
  )

```

```

(defrule parent6-icom-o
  (declare (salience -2))
  (parent6-boundary ?p6-name ?p6-data o)
  =>
    (assert (parent6-icom ?p6-name ?p6-data output 1))
  )

```

```

(defrule parent6-icom-i
  (declare (salience -2))
  (parent6-boundary ?p6-name ?p6-data i)
  =>
    (assert (parent6-icom ?p6-name ?p6-data input 1))
  )

```

```

(defrule parent6-icom-m
  (declare (salience -2))
  (parent6-boundary ?p6-name ?p6-data m)
  =>
    (assert (parent6-icom ?p6-name ?p6-data mech 1))
  )

```

```

; -----

```

```

(defrule parent6-control-add
  (declare (salience -3))
  ?f1<-(parent6-icom ?p6-name ?data1 control ?one)
  ?f2<-(parent6-icom ?p6-name ?data2 control ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent6-icom ?p6-name =(gensym) control ?total))
  )

```

; -----

```

(defrule parent6-output-add
  (declare (salience -3))
  ?f1<-(parent6-icom ?p6-name ?data1 output ?one)
  ?f2<-(parent6-icom ?p6-name ?data2 output ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent6-icom ?p6-name =(gensym) output ?total))
  )

```

; -----

```

(defrule parent6-input-add
  (declare (salience -3))
  ?f1<-(parent6-icom ?p6-name ?data1 input ?one)
  ?f2<-(parent6-icom ?p6-name ?data2 input ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent6-icom ?p6-name =(gensym) input ?total))
  )

```

```

(defrule parent6-mech-add
  (declare (salience -3))
  ?f1<-(parent6-icom ?p6-name ?data1 mech ?one)
  ?f2<-(parent6-icom ?p6-name ?data2 mech ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (parent6-icom ?p6-name =(gensym) mech ?total))
  )

```

; -----

```

(defrule child6-icom-c
  (declare (salience -2))
  (child6-boundary ?c6-parent ?c6-name ?c6-data c)
  =>
  (assert (child6-icom ?c6-parent ?c6-data control 1))
  )

(defrule child6-icom-o
  (declare (salience -2))
  (child6-boundary ?c6-parent ?c6-name ?c6-data o)
  =>
  (assert (child6-icom ?c6-parent ?c6-data output 1) )
  )

(defrule child6-icom-i
  (declare (salience -2))
  (child6-boundary ?c6-parent ?c6-name ?c6-data i)
  =>
  (assert (child6-icom ?c6-parent ?c6-data input 1) )
  )

(defrule child6-icom-m
  (declare (salience -2))
  (child6-boundary ?c6-parent ?c6-name ?c6-data m)
  =>
  (assert (child6-icom ?c6-parent ?c6-data mech 1) )
  )

; -----

(defrule child6-control-add
  (declare (salience -3))
  ?f1<-(child6-icom ?c6-parent ?data1 control ?one)
  ?f2<-(child6-icom ?c6-parent ?data2 control ?n)
  (test (neq ?data1 ?data2))
  =>
  (retract ?f1 ?f2)
  (bind ?total (+ ?one ?n))
  (assert (child6-icom ?c6-parent =(gensym) control ?total))
  )

; -----

(defrule child6-output-add
  (declare (salience -3))
  ?f1<-(child6-icom ?c6-parent ?data1 output ?one)
  ?f2<-(child6-icom ?c6-parent ?data2 output ?n)
  (test (neq ?data1 ?data2))

```

```

=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child6-icom ?c6-parent =(gensym) output ?total))
)

; -----

(defrule child6-input-add
(declare (salience -3))
?f1<-(child6-icom ?c6-parent ?data1 input ?one)
?f2<-(child6-icom ?c6-parent ?data2 input ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child6-icom ?c6-parent =(gensym) input ?total))
)
; -----

(defrule child6-mech-add
(declare (salience -3))
?f1<-(child6-icom ?c6-parent ?data1 mech ?one)
?f2<-(child6-icom ?c6-parent ?data2 mech ?n)
(test (neq ?data1 ?data2))
=>
(retract ?f1 ?f2)
(bind ?total (+ ?one ?n))
(assert (child6-icom ?c6-parent =(gensym) mech ?total))
)

;*****
;          Check Parent with 6 child  boundary icom number consistency
;-----

(defrule check-parent-6child-control
(declare (salience -6))
(parent6-icom ?p6-name ? control ?p)
(child6-icom ?p6-name ? control ?c)
(test (!= ?p ?c))
=>
(if (> ?p ?c)
then
(bind ?pd (- ?p ?c))
(printout t "WARNING, there might be an ERROR: The number of boundary controls" crlf)
(printout t "  of parent activity "?p6-name" is "?pd" control(s) more than " crlf)
(printout t "  its child activities." crlf)
(printout t "  Are there 'consists of' data items at boundary?" crlf)
)
)

```

```

(printout t " Please recheck the syntax." crlf)
else
(bind ?cd (- ?c ?p))
(printout t "WARNING, there might be an ERROR: The number of boundary controls" crlf)
(printout t " of the parent activity "?p6-name" is "?cd" control(s) less " crlf)
(printout t " than its child boundary controls." crlf)
(printout t " Are there 'consists of' data items at boundary?" crlf)
(printout t " Please recheck the syntax." crlf)
) )

```

```

(defrule check-parent-6child-output
(declare (salience -6))
(parent6-icom ?p6-name ? output ?p)
(child6-icom ?p6-name ? output ?c)
(test (!= ?p ?c))
=>
(if (> ?p ?c)
then
(bind ?pd (- ?p ?c))
(printout t "WARNING, there might be an ERROR: The number of boundary outputs" crlf)
(printout t " of parent activity " ?p6-name " is " ?pd " output(s) more " crlf)
(printout t " than its child activities." crlf)
(printout t " Are there 'consists of' data items at boundary?" crlf)
(printout t " Please recheck the syntax." crlf)
else
(bind ?cd (- ?c ?p))
(printout t "WARNING, there might be an ERROR: The number of boundary outputs" crlf)
(printout t " of the parent activity " ?p6-name " is " ?cd " output(s) less " crlf)
(printout t " than its child boundary outputs." crlf)
(printout t " Are there 'consists of' data items at boundary?" crlf)
(printout t " Please recheck the syntax." crlf)
) )

```

```

(defrule check-parent-6child-input
(declare (salience -6))
(parent6-icom ?p6-name ? input ?p)
(child6-icom ?p6-name ? input ?c)
(test (!= ?p ?c))
=>
(if (> ?p ?c)
then
(bind ?pd (- ?p ?c))
(printout t "WARNING, there might be an ERROR: The number of boundary inputs" crlf)
(printout t " of parent activity " ?p6-name " is " ?pd " input(s) more " crlf)
(printout t " than its child activities." crlf)
(printout t " Are there 'consists of' data items at boundary?" crlf)
(printout t " Please recheck the syntax." crlf)
else
(bind ?cd (- ?c ?p))
(printout t "WARNING, there might be an ERROR: The number of boundary inputs" crlf)

```

```

(printout t " of the parent activity " ?p6-name " is " ?cd " input(s) less " crlf)
(printout t " than its child boundary inputs." crlf)
(printout t " Are there 'consists of' data items at boundary?" crlf)
(printout t " Please recheck the syntax." crlf)
) )

```

```

(defrule check-parent-6child-mech
  (declare (salience -6))
  (parent6-icom ?p6-name ? mech ?p)
  (child6-icom ?p6-name ? mech ?c)
  (test (≠ ?p ?c))
  =>
  (if (> ?p ?c)
    then
    (bind ?pd (- ?p ?c))
    (printout t "WARNING, there might be an ERROR: The number of boundary mechanisms" crlf)
    (printout t " of parent activity " ?p6-name " is " ?pd " mechanism(s) more " crlf)
    (printout t " than its child activities." crlf)
    (printout t " Are there 'consists of' data items at boundary?" crlf)
    (printout t " Please recheck the syntax." crlf)
    else
    (bind ?cd (- ?c ?p))
    (printout t "WARNING, there might be an ERROR: The number of boundary mechanisms" crlf)
    (printout t " of the parent activity " ?p6-name " is " ?cd " mechanism(s) less " crlf)
    (printout t " its child child boundary mechanisms." crlf)
    (printout t " Are there 'consists of' data items at boundary?" crlf)
    (printout t " Please recheck the syntax." crlf)
  ) )

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      Auxiliary Rules for checking any parent activity that have more than
;;      six child activities up to 3 levels of hierarchy.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defrule create-A7
  (declare (salience 5))
  (act-numb ?acta7 A7)
  =>
  (assert (act ?acta7 7))
)

```

```

(defrule create-A17
  (declare (salience 5))
  (act-numb ?acta17 A17)
  =>
  (assert (act ?acta17 1 7))
)

```

```

(defrule create-A27

```



```

    (declare (salience 5))
    (act-numb ?acta27 A27)
=>
    (assert (act ?acta27 2 7))
)

(defrule create-A37
  (declare (salience 5))
  (act-numb ?acta37 A37)
=>
  (assert (act ?acta37 3 7 ))
)

(defrule create-A47
  (declare (salience 5))
  (act-numb ?acta47 A47)
=>
  (assert (act ?acta47 4 7))
)

(defrule create-A57
  (declare (salience 5))
  (act-numb ?acta57 A57)
=>
  (assert (act ?acta57 5 7))
)

(defrule create-A67
  (declare (salience 5))
  (act-numb ?acta67 A67)
=>
  (assert (act ?acta67 6 7))
)

(defrule create-A117
  (declare (salience 5))
  (act-numb ?acta117 A117)
=>
  (assert (act ?acta117 1 1 7))
)

(defrule create-A127
  (declare (salience 5))
  (act-numb ?acta127 A127)
=>
  (assert (act ?acta127 1 2 7))
)

(defrule create-A137
  (declare (salience 5))
  (act-numb ?acta137 A137)
=>

```

```
(assert (act ?acta137 1 3 7))  
)
```

```
(defrule create-A147  
  (declare (salience 5))  
  (act-numb ?acta147 A147 )  
=>  
  (assert (act ?acta147 1 4 7))  
)
```

```
(defrule create-A157  
  (declare (salience 5))  
  (act-numb ?acta157 A157)  
=>  
  (assert (act ?acta157 1 5 7))  
)
```

```
(defrule create-A167  
  (declare (salience 5))  
  (act-numb ?acta167 A167)  
=>  
  (assert (act ?acta167 1 6 7))  
)
```

```
(defrule create-A217  
  (declare (salience 5))  
  (act-numb ?acta217 A217)  
=>  
  (assert (act ?acta217 2 1 7))  
)
```

```
(defrule create-A227  
  (declare (salience .))  
  (act-numb ?acta227 A227)  
=>  
  (assert (act ?acta227 2 2 7))  
)
```

```
(defrule create-A237  
  (declare (salience 5))  
  (act-numb ?acta237 A237)  
=>  
  (assert (act ?acta237 2 3 7))  
)
```

```
(defrule create-A247  
  (declare (salience 5))  
  (act-numb ?acta247 A247)  
=>  
  (assert (act ?acta247 2 4 7))  
)
```

```

(defrule create-A257
  (declare (salience 5))
  (act-numb ?acta257 A257)
=>
  (assert (act ?acta257 2 5 7))
)

```

```

(defrule create-A267
  (declare (salience 5))
  (act-numb ?acta267 A267)
=>
  (assert (act ?acta267 2 6 7))
)

```

```

(defrule create-A317
  (declare (salience 5))
  (act-numb ?acta317 A317)
=>
  (assert (act ?acta317 3 1 7))
)

```

```

(defrule create-A327
  (declare (salience 5))
  (act-numb ?acta327 A327)
=>
  (assert (act ?acta327 3 2 7))
)

```

```

(defrule create-A337
  (declare (salience 5))
  (act-numb ?acta337 A337)
=>
  (assert (act ?acta337 3 3 7))
)

```

```

(defrule create-A347
  (declare (salience 5))
  (act-numb ?acta347 A347)
=>
  (assert (act ?acta347 3 4 7))
)

```

```

(defrule create-A357
  (declare (salience 5))
  (act-numb ?acta357 A357)
=>
  (assert (act ?acta357 3 5 7))
)

```

```

(defrule create-A367
  (declare (salience 5))
  (act-numb ?acta367 A367)
)

```

```

=>
  (assert (act ?acta367 3 6 7))
)

(defrule create-A417
  (declare (salience 5))
  (act-numb ?acta417 A417)
=>
  (assert (act ?acta417 4 1 7))
)
(defrule create-A427
  (declare (salience 5))
  (act-numb ?acta427 A427)
=>
  (assert (act ?acta427 4 2 7))
)

(defrule create-A437
  (declare (salience 5))
  (act-numb ?acta437 A437)
=>
  (assert (act ?acta437 4 3 7))
)

(defrule create-A447
  (declare (salience 5))
  (act-numb ?acta447 A447)
=>
  (assert (act ?acta447 4 4 7))
)

(defrule create-A457
  (declare (salience 5))
  (act-numb ?acta457 A457)
=>
  (assert (act ?acta457 4 5 7))
)

(defrule create-A467
  (declare (salience 5))
  (act-numb ?acta467 A467)
=>
  (assert (act ?acta467 4 6 7))
)

(defrule create-A517
  (declare (salience 5))
  (act-numb ?acta517 A517)
=>
  (assert (act ?acta517 5 1 7))
)

```

```

(defrule create-A527
  (declare (salience 5))
  (act-numb ?acta527 A527)
=>
  (assert (act ?acta527 5 2 7))
)
(defrule create-A537
  (declare (salience 5))
  (act-numb ?acta537 A537)
=>
  (assert (act ?acta537 5 3 7))
)
(defrule create-A547
  (declare (salience 5))
  (act-numb ?acta547 A547)
=>
  (assert (act ?acta547 5 4 7))
)
(defrule create-A557
  (declare (salience 5))
  (act-numb ?acta557 A557)
=>
  (assert (act ?acta557 5 5 7))
)
(defrule create-A567
  (declare (salience 5))
  (act-numb ?acta567 A567)
=>
  (assert (act ?acta567 5 6 7))
)
(defrule create-A617
  (declare (salience 5))
  (act-numb ?acta617 A617)
=>
  (assert (act ?acta617 6 1 7))
)
(defrule create-A627
  (declare (salience 5))
  (act-numb ?acta627 A627)
=>
  (assert (act ?acta627 6 2 7))
)
(defrule create-A637
  (declare (salience 5))
  (act-numb ?acta637 A637)

```

```
=>
  (assert (act ?acta637 6 3 7))
)
```

```
(defrule create-A647
  (declare (salience 5))
  (act-numb ?acta647 A647)
=>
  (assert (act ?acta647 6 4 7))
)
```

```
(defrule create-A657
  (declare (salience 5))
  (act-numb ?acta657 A657)
=>
  (sert (act ?acta657 6 5 7))
)
```

```
(defrule create-A667
  (declare (salience 5))
  (act-numb ?acta667 A667)
=>
  (assert (act ?acta667 6 6 7))
)
```

```
;;;;;;;;;;;;;; END OF SATool II RULE BASE ;;;;;;;;;;;;;;
```

Appendix D. *SAMPLE ESSENTIAL MODEL IDEF₀ SYNTAX CHECKING*

SCRIPT FILE

NOTICE: Any comments added by the author will be followed by a ';'.
;

csh> a.out

CLIPS/Ada Version 4.30 10/12/89

```
*****
*           THE ESSENTIAL SUBSYSTEM           *
*           TEST AND DEMONSTRATION MAIN MENU   *
*           -- SATool Il Level Operations --   *
*****
```

```
Enter      To select the desired operation
  1.      Restore (load) a project from disk
          (Warning: all current data cleared)
  2.      Save the current project to disk
  3.      Display the current project name
  4.      Change the current project name
  5.      Create and display a data dictionary entry
  6.      Add a box/activity to the project
  7.      Connect 2 boxes with a data element/arrow
  8.      Check Syntax of current project
  9.      -- Submenus for Low Level Operations --
  0.      EXIT
```

SELECT A NUMBER: 1

Enter the file name of the project to be restored.

Do not include the file name extension.

Enter Name: thesis_err

Looking for essential data under file name: thesis_err.esm

Preparing to read facts from disk into a buffer.

A set of facts has been extracted from the file.

Calling procedure to load icom facts.

Procedure to restore ICOM facts done.

A set of facts has been extracted from the file.

Calling procedure to load project name fact.

Procedure to restore project name is done.

A set of facts has been extracted from the file.

Calling procedure to load activity facts.

Procedure to restore activity facts is done.

A set of facts has been extracted from the file.

Calling procedure to load data element facts.

Procedure to restore data element facts is done.

A set of facts has been extracted from the file.

Calling procedure to load historical activity facts.

Procedure to restore historical activity facts is done.
 A set of facts has been extracted from the file.
 Calling procedure to load calls relation facts.
 Procedure to restore calls relation facts is done.
 A set of facts has been extracted from the file.
 Calling procedure to load consists of relation facts.
 Procedure to restore consists of relation facts is done.
 Project successfully restored.

PRESS ANY KEY - THEN RETURN TO CONTINUE: 1

```

*****
*           THE ESSENTIAL SUBSYSTEM           *
*       TEST AND DEMONSTRATION MAIN MENU       *
*       -- SATool II Level Operations --       *
*****
  
```

```

Enter    To select the desired operation
  1.      Restore (load) a project from disk
          (Warning: all current data cleared)
  2.      Save the current project to disk
  3.      Display the current project name
  4.      Change the current project name
  5.      Create and display a data dictionary entry
  6.      Add a box/activity to the project
  7.      Connect 2 boxes with a data element/arrow
  8.      Check Syntax of current project
  9.      -- Submenus for Low Level Operations --
  0.      EXIT
  
```

SELECT A NUMBER: 9

```

*****
* ESSENTIAL SUBSYSTEM EDITING AND DEBUGGING MENU *
* Warning: These operations allow you to directly *
* exercise the object operations. Use extreme care.*
* --Essential Model and Utility Level Operations-- *
*****
  
```

```

Enter    To select the desired submenu of operations
  1.      Activity Operations Menu
  2.      Data Element Operations Menu
  3.      Historical Activity Operations Menu
  4.      Calls Relation Operations Menu
  5.      ICOM Relation Operations Menu

  6.      Consists_Of Relation Operations Menu
  7.      CLIPS Operations Menu
  8.      ICOM Fact Operations Menu
  9.      Activity Fact Operations Menu
  0.      EXIT
  
```

SELECT A NUMBER: 7

Enter To select this operation


```

1. Assert all facts into the CLIPS Working Memory.
2. Display all the facts in CLIPS Working Memory.
3. Clear the CLIPS Working Memory.
0. EXIT
SELECT A NUMBER: 1
ICOM facts for CLIPS retrieved, if any.
CLIPS WM - a set of facts were asserted.
Project name fact retrieved.
CLIPS WM - a set of facts were asserted.
Activity facts for CLIPS retrieved, if any.
CLIPS WM - a set of facts were asserted.
Data element facts for CLIPS retrieved, if any.
CLIPS WM - a set of facts were asserted.
Historical facts for CLIPS retrieved, if any.
CLIPS WM - a set of facts were asserted.
Calls relation facts for CLIPS retrieved, if any.
CLIPS WM - a set of facts were asserted.
Consists of relation facts for CLIPS retrieved, if any.
CLIPS WM - a set of facts were asserted.
All facts for CLIPS retrieved.

```

PRESS ANY KEY - THEN RETURN TO CONTINUE: 1

Enter To select this operation

```

1. Assert all facts into the CLIPS Working Memory.
2. Display all the facts in CLIPS Working Memory.
3. Clear the CLIPS Working Memory.
0. EXIT
SELECT A NUMBER: 2

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Notice: Each fact is assigned a fact number, f-#.
;   There are 753 facts. Only one set of facts are kept here
; as an example.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

*****Start of Working Memory*****.
f-1    (icom-tuple Control_Elevator summons_indication c 1)
f-2    (icom-tuple Control_Elevator floor_sensor c 2)
f-3    (icom-tuple Control_Elevator door_sensor c 3)
f-4    (icom-tuple Control_Elevator system_control c 4)
f-5    (icom-tuple Control_Elevator control_signals o 5)
f-6    (icom-tuple Control_Elevator passenger_requests i 7)
f-7    (icom-tuple Control_Elevator overload_sensor i 8)
f-8    (icom-tuple Control_Elevator floor_motor_drive m 9)
f-9    (icom-tuple Control_Elevator door_motor_drive m 10.99999999)
.....
f-74   (icom-activity-inputs Control_Elevator 2)
f-75   (icom-activity-controls Control_Elevator 4)
f-76   (icom-activity-outputs Control_Elevator 1)

```

```

f-77  (icom-activity-mechanisms Control_Elevator 2)
.....
f-158  (project-name Control_Elevator)
f-159  (act-name Control_Elevator)
f-160  (act-numb Control_Elevator A0)
f-161  (act-desc Control_Elevator not-null)
f-162  (act-has-child Control_Elevator Store_Request)
f-163  (act-has-child Control_Elevator Elevator_Control)
f-164  (act-has-child Control_Elevator Schedule_Elevator)
f-165  (act-ref-type Control_Elevator null)
f-166  (act-ref Control_Elevator null)
f-167  (act-version Control_Elevator null)
f-168  (act-ver-chg Control_Elevator null)
f-169  (act-date Control_Elevator null)
f-170  (act-author Control_Elevator null)
.....
f-384  (data-element-name summons_indication)
f-385  (data-element-type summons_indication null)
f-386  (data-element-minimum summons_indication null)
f-387  (data-element-maximum summons_indication null)
f-388  (data-element-data-range summons_indication null)
f-389  (data-element-values summons_indication null)
f-390  (data-desc summons_indication not-null)
f-391  (data-ref summons_indication null)
f-392  (data-ref-type summons_indication null)
f-393  (data-ele-ver summons_indication null)
f-394  (data-e-v-chg summons_indication null)
f-395  (data-ele-date summons_indication null)
f-396  (data-ele-author summons_indication null)
.....
f-748  (historical-tuple Control_Elevator A0)
f-749  (calls-relation-tuple Building_Transport Building_Structure A11)
f-750  (consists-of-name 1 elevator_status up/down)
f-751  (consists-of-name 2 elevator_status stopped)
f-752  (consists-of-name 6 sequenced_signals earlier_signals)
f-753  (consists-of-name 7 sequenced_signals later_signals)
*****End of Working Memory*****

```

PRESS ANY KEY - THEN RETURN TO CONTINUE: 1

Enter To select this operation

1. Assert all facts into the CLIPS Working Memory.
2. Display all the facts in CLIPS Working Memory.
3. Clear the CLIPS Working Memory.
0. EXIT

SELECT A NUMBER: 0

PRESS ANY KEY - THEN RETURN TO CONTINUE: 1

```

*****
* ESSENTIAL SUBSYSTEM EDITING AND DEBUGGING MENU *
* Warning: These operations allow you to directly *
* exercise the object operations. Use extreme care.*
* --Essential Model and Utility Level Operations-- *
*****

```

```

Enter    To select the desired submenu of operations
1.       Activity Operations Menu
2.       Data Element Operations Menu
3.       Historical Activity Operations Menu
4.       Calls Relation Operations Menu
5.       ICOM Relation Operations Menu
6.       Consists_Of Relation Operations Menu
7.       CLIPS Operations Menu
8.       ICOM Fact Operations Menu
9.       Activity Fact Operations Menu
0.       EXIT

```

SELECT A NUMBER: 0

PRESS ANY KEY - THEN RETURN TO CONTINUE: 1

```

*****
* THE ESSENTIAL SUBSYSTEM *
* TEST AND DEMONSTRATION MAIN MENU *
* -- SAtool II Level Operations -- *
*****

```

```

Enter    To select the desired operation
1.       Restore (load) a project from disk
          (Warning: all current data cleared)
2.       Save the current project to disk
3.       Display the current project name
4.       Change the current project name
5.       Create and display a data dictionary entry
6.       Add a box/activity to the project
7.       Connect 2 boxes with a data element/arrow
8.       Check Syntax of current project
9.       -- Submenus for Low Level Operations --
0.       EXIT

```

SELECT A NUMBER: 8

ICOM facts for CLIPS retrieved, if any.
 CLIPS WM - a set of facts were asserted.
 Project name fact retrieved.
 CLIPS WM - a set of facts were asserted.
 Activity facts for CLIPS retrieved, if any.
 CLIPS WM - a set of facts were asserted.
 Data element facts for CLIPS retrieved, if any.
 CLIPS WM - a set of facts were asserted.
 Historical facts for CLIPS retrieved, if any.
 CLIPS WM - a set of facts were asserted.
 Calls relation facts for CLIPS retrieved, if any.

CLIPS WM - a set of facts were asserted.
Consists of relation facts for CLIPS retrieved, if any.
CLIPS WM - a set of facts were asserted.

**** Essential Subsystem Syntax Checking Messages ****

==> The project == Control_Elevator == is checked as follows:

```
;;;;;;;;;;;;;
;   ERRORS are related. If a parent activity can't find
;   a boundary arrow in its child activities boundaries, that
;   means its child activities can't also find a boundary arrow
;   from its parent activity. Two ERRORS will be raised
;   for both the parent and child activities. Thus, a middle
;   level data error will raise 4 ERROR messages. Since there
;   are two sets of parent and child activities.
;
;   A set of same IDEFO figures as described in chapter II
;   MARKED with DESIGNED ERRORS is listed below. Inconsistent
;   data input are marked with a no, not or a false
;   preceding the data name.
;;;;;;;;;;;;;
```

**** Essential Subsystem Syntax Checking Messages ****

==> The project == Control_Elevator == is checked as follows:

Warning: activity A2 has more than 6 child diagrams.
Notice: Please manually check to make sure that there is no
such an warning lower than 4 levels of hierarchy.
; Activity A27

WARNING: Activity number A265 Send_Signals needs a
description.
ERROR: Activity Check_Destination needs at least 1 control.
; Activity A21 should'nt have two input

ERROR: Data inconsistency between parent activity
Sort_Signals data 'o' false_signals and its child
diagrams.
; Parent A26 output

ERROR: Data inconsistency between child activity
Send_Signals data 'o' signals and its
parent.
; Child A26 output

ERROR: Data inconsistency between child activity
Compare_Signals data 'c' not_confirmed and its
parent.

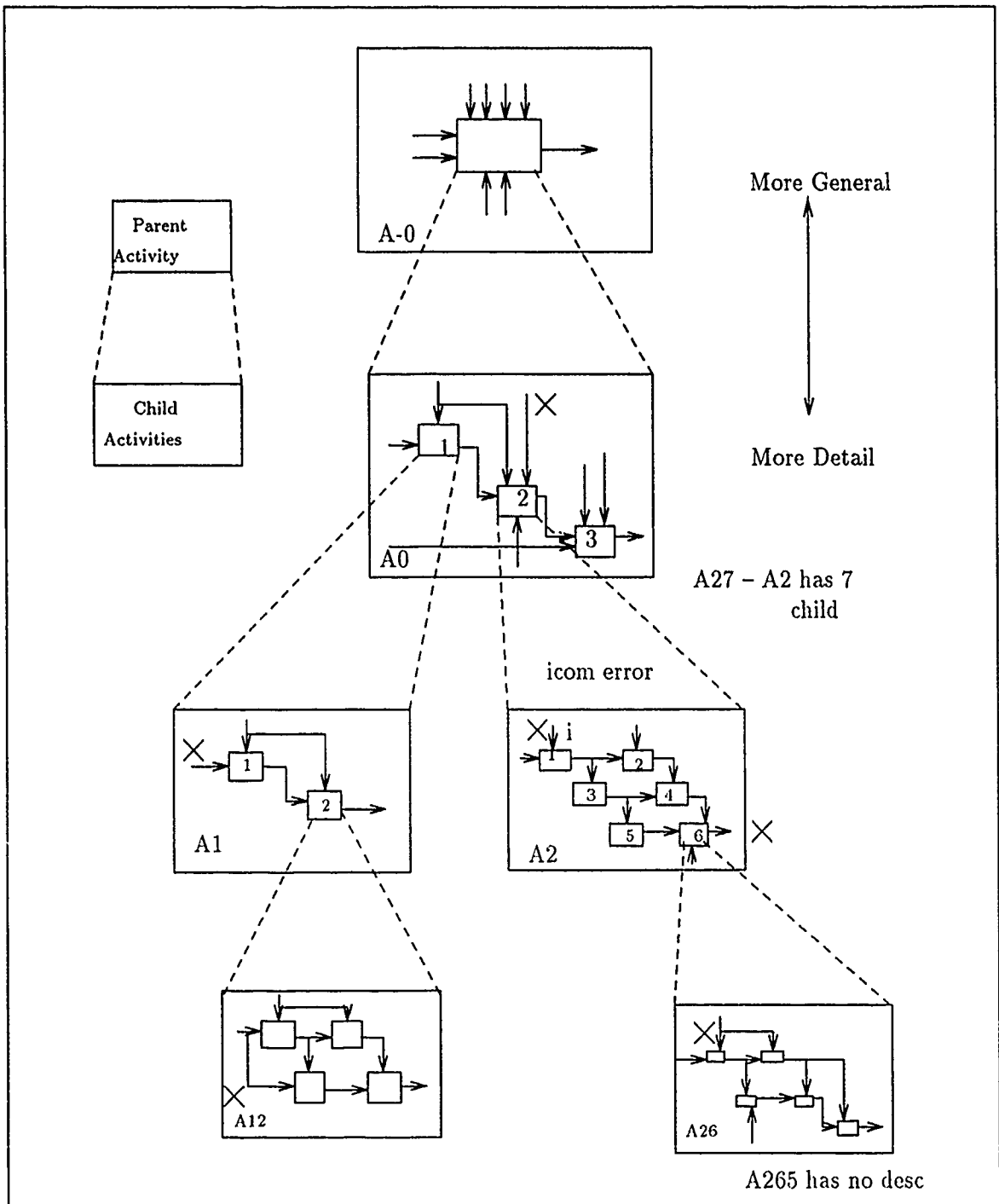


Figure D.1. Hierarchy Diagram for 'Control Elevator'

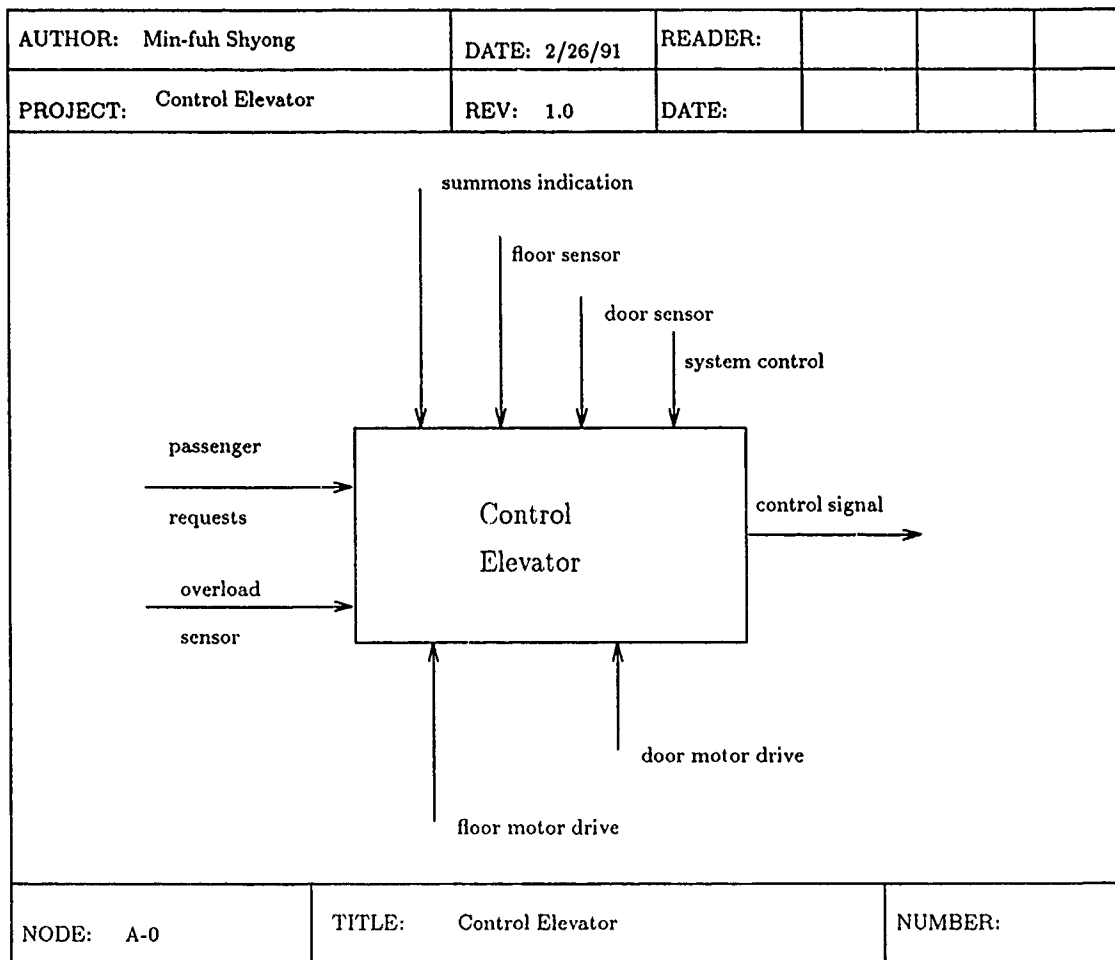


Figure D.2. A-0 Essential Model Diagram for 'Control Elevator'

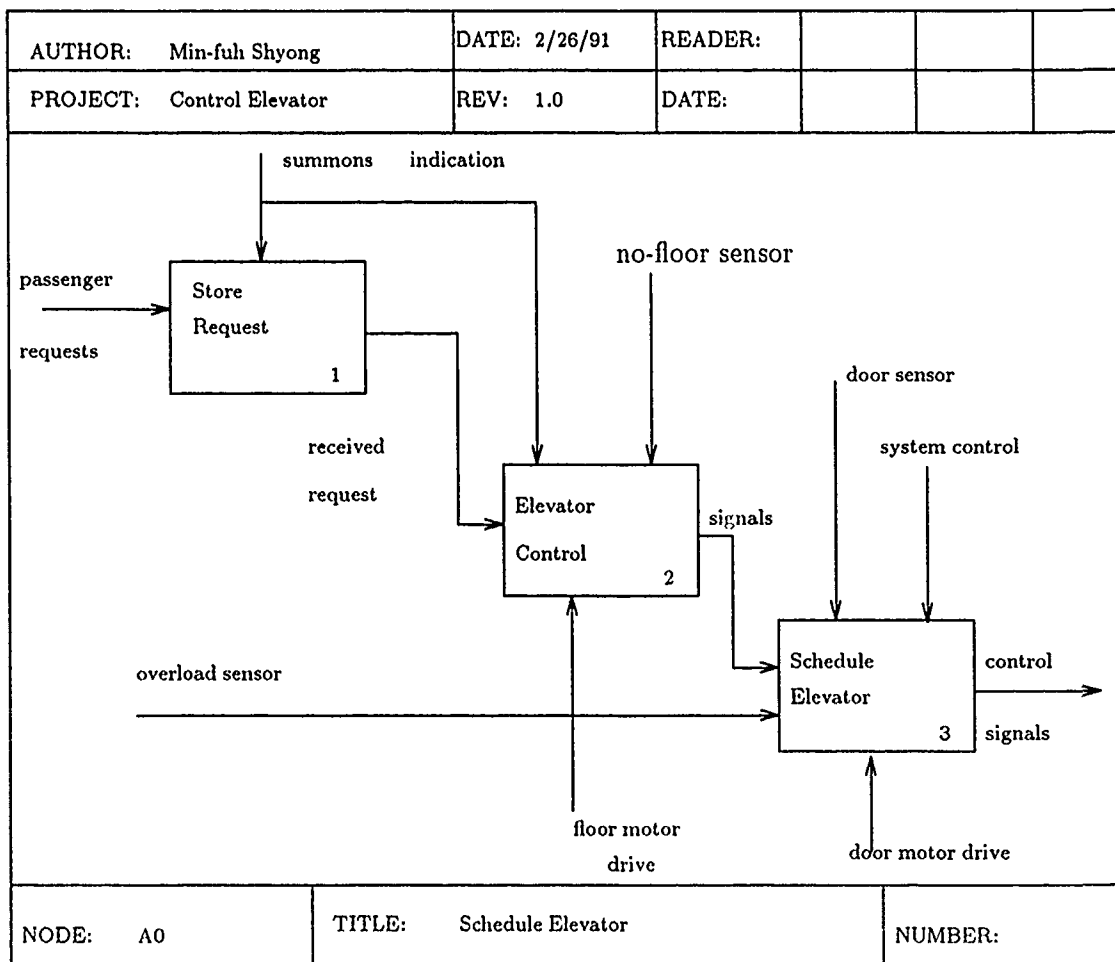


Figure D.3. A0 Diagram for 'Control Elevator'

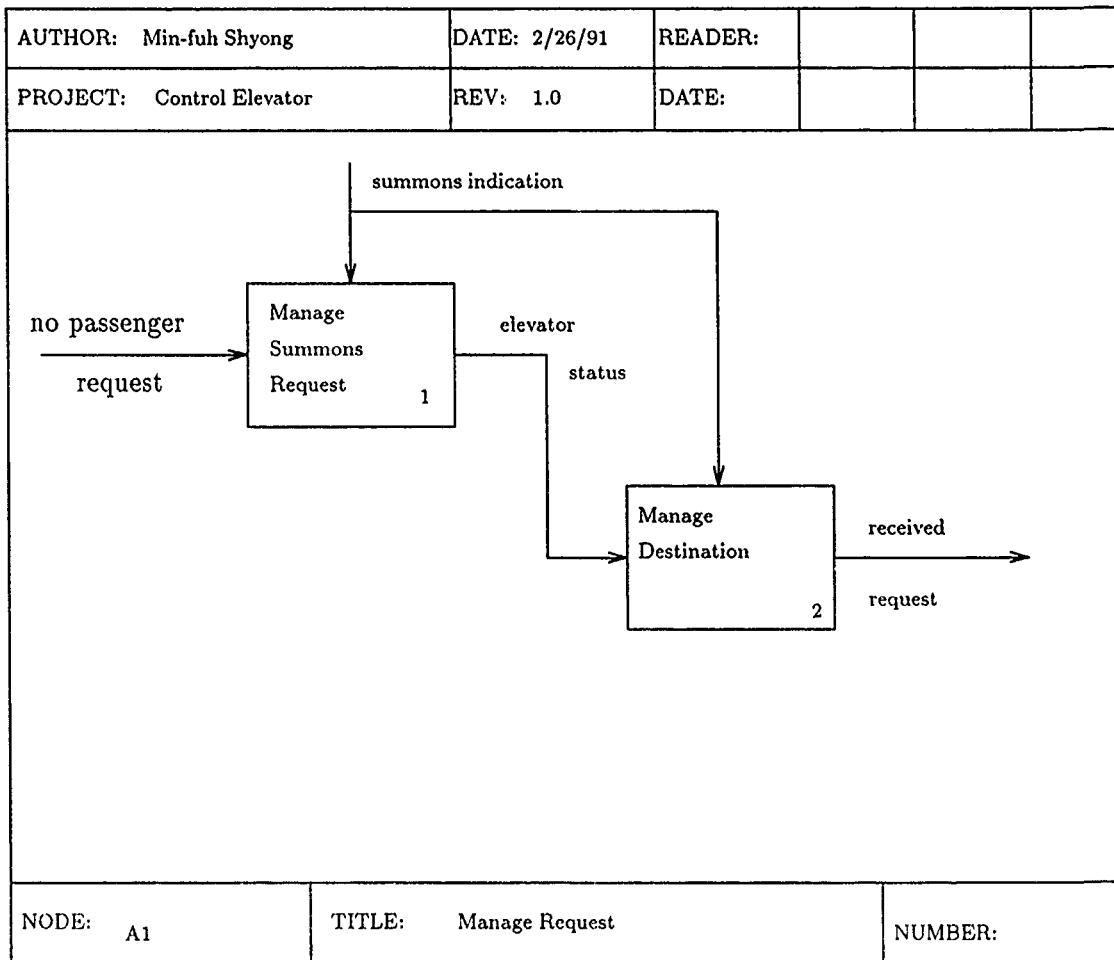


Figure D.4. A1 Diagram for 'Control Elevator'

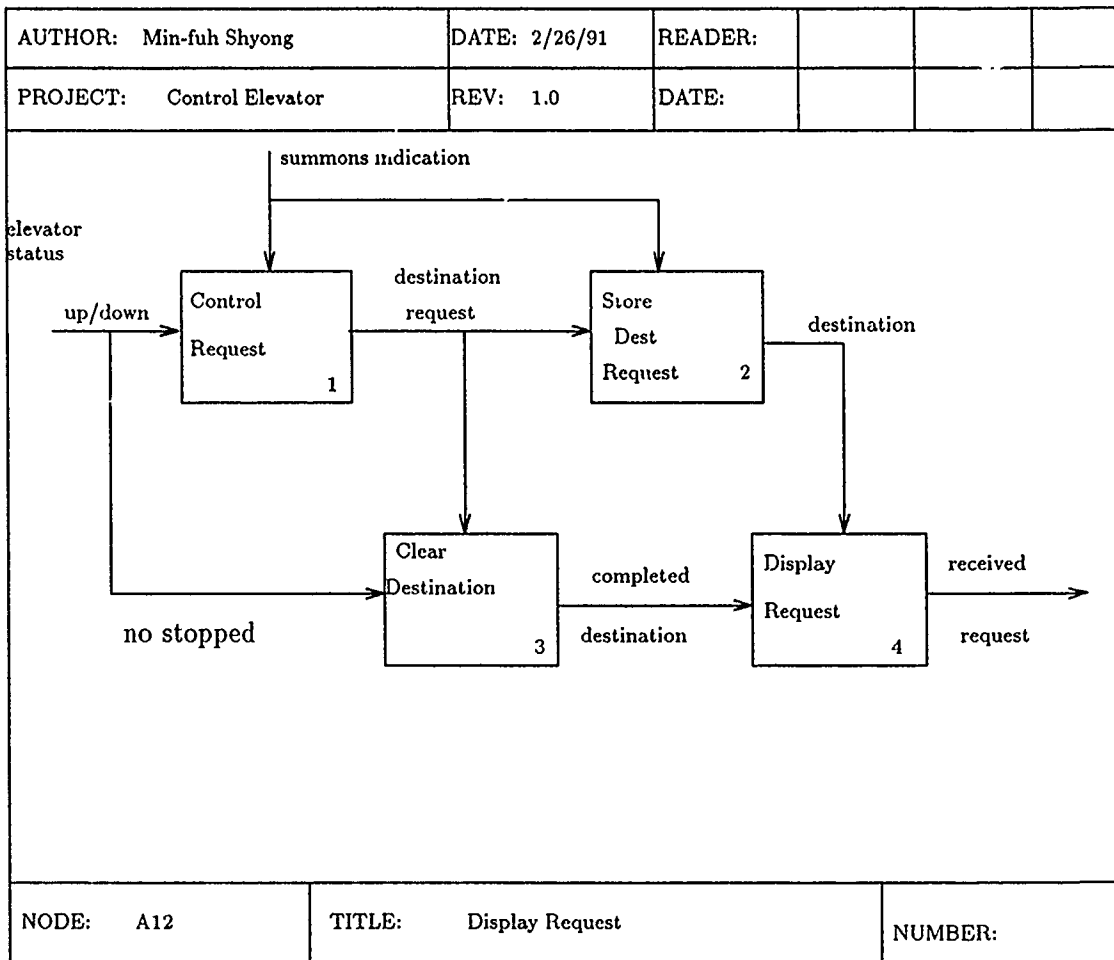


Figure D.5. A12 Diagram for 'Control Elevator'

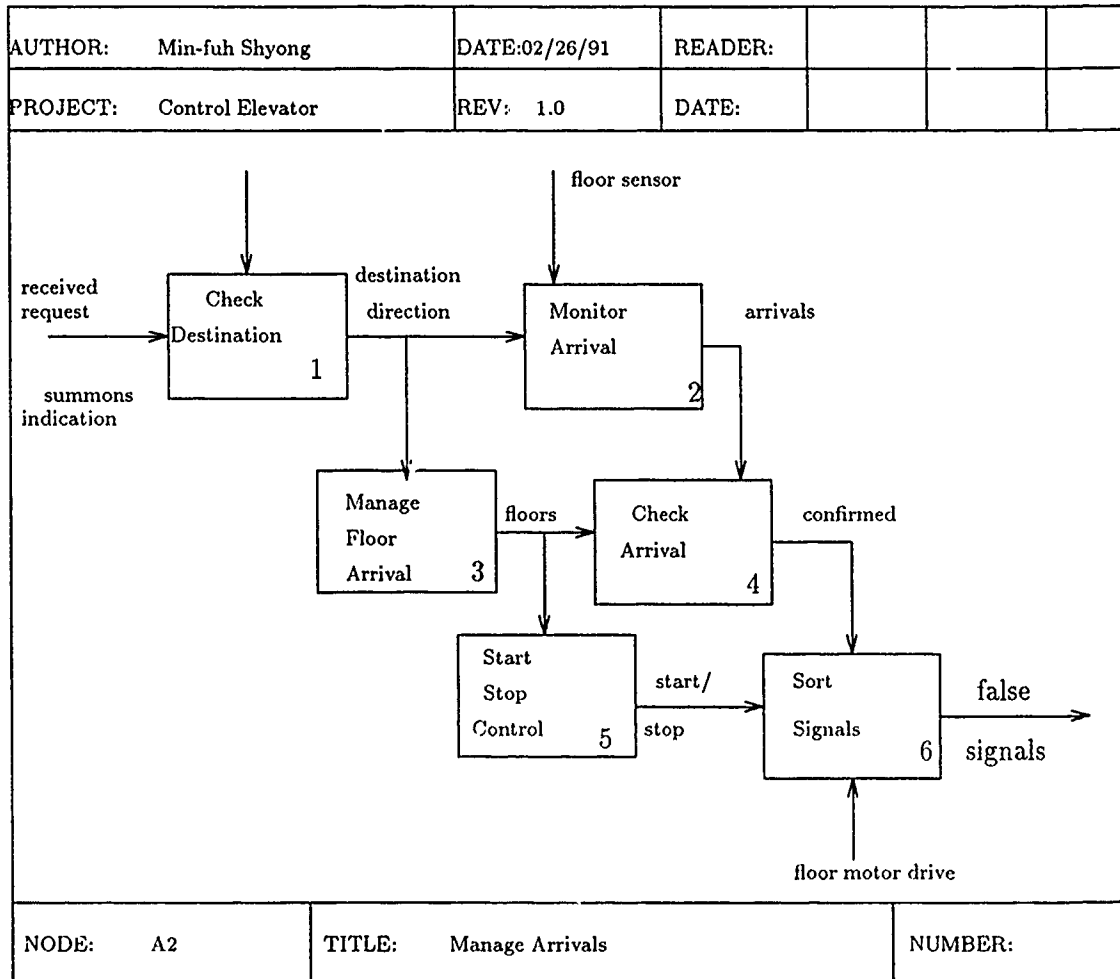


Figure D.6. A2 Diagram for 'Control Elevator'

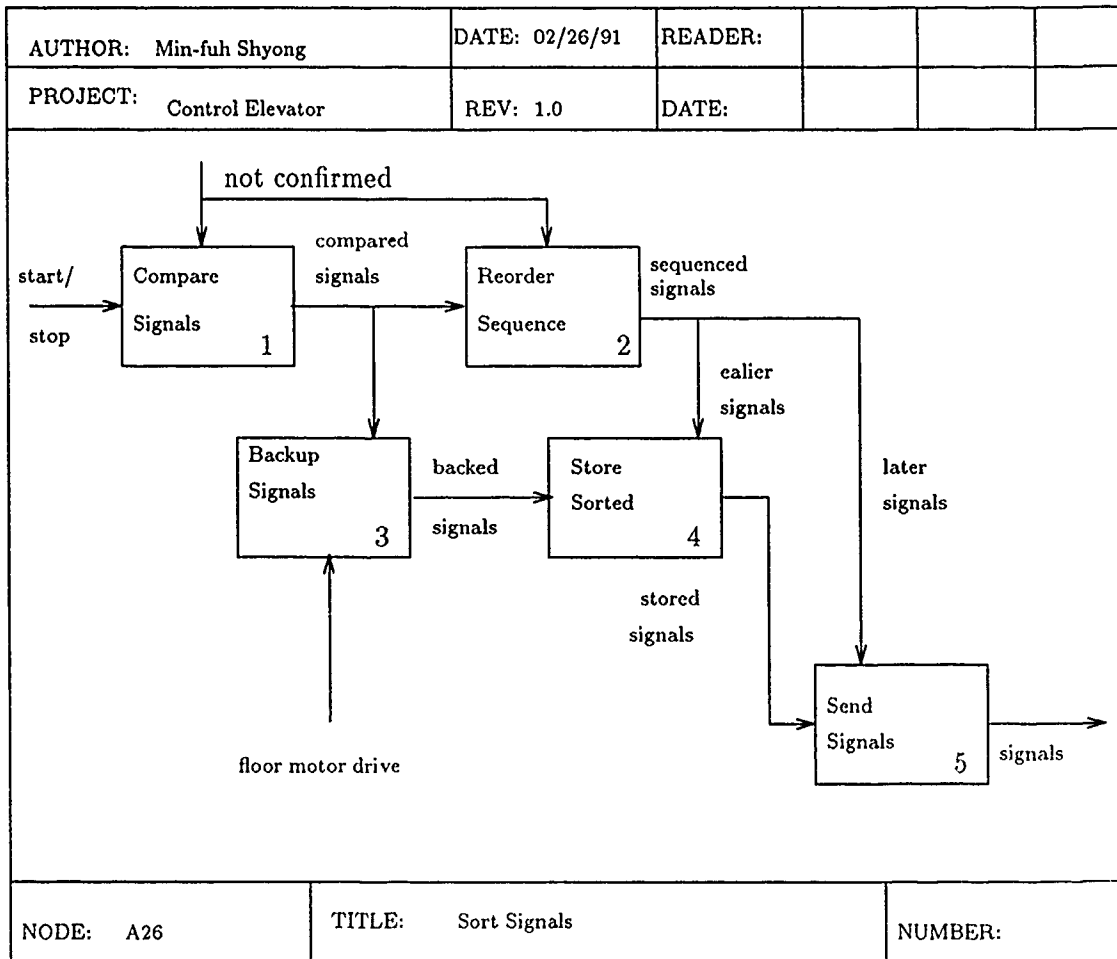


Figure D.7. A26 Diagram for 'Control Elevator'

```

; Child A231 control

ERROR: Data inconsistency between child activity
Clear_Destination data 'i' no_stopped and its
parent.
; Child of A12, A123 pipelined input

ERROR: Data inconsistency between parent activity
Elevator_Control data 'o' signals and its child
diagrams.
; Parent A2 output and A26 output

ERROR: Data inconsistency between parent activity
Elevator_Control data 'c' no_floor_sensor and its child
diagrams.
; Parent A2

ERROR: icom inconsistency between activity Elevator_Control and its
child diagram Check_Destination.
; Parent A2 icom inconsistent with its child

ERROR: Data inconsistency between child activity
Sort_Signals data 'o' false_signals and its
parent.
; A26 output inconsistent with A2 output

ERROR: Data inconsistency between child activity
Monitor_Arrival data 'c' floor_sensor and its
parent.
; A22 control inconsistent with A2

ERROR: Data inconsistency between parent activity
Store_Request data 'i' passenger_requests and its
child diagrams.
; A1 input inconsistent with A11

ERROR: Data inconsistency between parent activity
Control_Elevator data 'c' floor_sensor and its child
diagrams.
; Parent A0 control - A2

ERROR: Data inconsistency between child activity
Elevator_Control data 'c' no_floor_sensor and its
parent.
; Child A2 control - A0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Since there are ERRORS occurred, so the icom number
; between each pair of parent and child activities (no matter
; at what level) might be inconsistent. But it might because
; of a pipeline data element. So a WARNING will be fired

```

```

; by the syntax checking expert system.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

WARNING, there might be an ERROR: The number of boundary controls
of the parent activity Sort_Signals is 1 control(s) less
than its child boundary controls.
Are there 'consists of' data items at boundary?
Please recheck the syntax.
WARNING, there might be an ERROR: The number of boundary inputs
of the parent activity Manage_Destination is 1 input(s) less
than its child boundary inputs.
Are there 'consists of' data items at boundary?
Please recheck the syntax.
WARNING, there might be an ERROR: The number of boundary controls
of parent activity Elevator_Control is 1 control(s) more than
its child activities.
Are there 'consists of' data items at boundary?
Please recheck the syntax.
WARNING, there might be an ERROR: The number of boundary inputs
of the parent activity Elevator_Control is 1 input(s) less
than its child boundary inputs.
Are there 'consists of' data items at boundary?
Please recheck the syntax.
ERROR: Data inconsistency between parent activity
Manage_Destination data 'i' elevator_status and its child
diagrams.
; Parent A12 pipeline data outout, child data inconsistent

Clips run completed. Rules fired = 196

PRESS ANY KEY - THEN RETURN TO CONTINUE:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; NOTE: If no errors were found; only a few 'consists of' data ;
; elements are used at boundary. Then the Syntax Expert System ;
; will give the following checking results. ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

**** Essential Subsystem Syntax Checking Messages ****
==> The project == Control_Elevator == is checked as follows:

WARNING, there might be an ERROR: The number of boundary inputs
of the parent activity Manage_Destination is 1 input(s) less
than its child boundary inputs.
Are there 'consists of' data items at boundary?
Please recheck the syntax.
CONGRATULATIONS: No syntax errors encountered.
SUGGESTION: Please recheck logical structure of your project
for perfection

```

Clips run completed. Rules fired = 175

PRESS ANY KEY - THEN RETURN TO CONTINUE: 1

```
*****
*           THE ESSENTIAL SUBSYSTEM           *
*           TEST AND DEMONSTRATION MAIN MENU    *
*           -- SAtool II Level Operations --    *
*****
```

```
Enter      To select the desired operation
1.         Restore (load) a project from disk
           (Warning: all current data cleared)
2.         Save the current project to disk
3.         Display the current project name
4.         Change the current project name
5.         Create and display a data dictionary entry
6.         Add a box/activity to the project
7.         Connect 2 boxes with a data element/arrow
8.         Check Syntax of current project
9.         -- Submenus for Low Level Operations --
0.         EXIT
```

SELECT A NUMBER: 0

csh> exit

csh>